

S

S@n

► Success at n

Safety and Domain Independence

RODNEY TOPOR

Griffith University, Nathan, QLD, Australia

Synonyms

Finiteness

Definition

The values in the relations of a relational database are elements of one or more underlying sets called domains. In practical applications, a domain may be infinite, e.g., the set of natural numbers. In this case, the value of a relational calculus query when applied to such a database may be infinite, e.g., $\{n \mid n \geq 10\}$. A query Q is called *finite* if the value of Q when applied to any database is finite.

Even when the database domains are finite, all that is normally known about them is that they are some finite superset of the values that occur in the database. In this case, the value of a relational calculus query may depend on such an unknown domain, e.g., $\{x \mid \forall y R(x, y)\}$. A query Q is called *domain independent* if the value of Q when applied to any database is the same for any two domains containing the database values or, equivalently, if the value of Q when applied to a database contains only values that occur in the database.

The term *safe* query has been used ambiguously in the literature. Often safe queries have been identified with finite queries. Sometimes safe queries have been members of a large, simple, decidable class of queries that are guaranteed to be finite, or, in other cases, domain independent. The use of word *safe* is preferred to denote a large, simple, decidable class of queries

that are guaranteed to be domain independent and hence, normally finite.

Obviously, it is desirable that queries be finite and domain independent. Unfortunately, the classes of finite queries and domain independent queries are undecidable, which leads to a search for decidable classes of queries that can represent all, or as many as possible, finite (resp., domain independent) queries.

Historical Background

DiPaola [3] and independently Vardi [11] recognized the desirability that queries be domain independent and proved that the class of domain independent queries was undecidable.

Many researchers then attempted to define decidable classes of queries that were guaranteed to be domain independent. This work was summarized by Topor [9], Kifer [6], Ullman [10], and Abiteboul *et al.* [1]. Many different names such as range-restricted, allowed, safe, with subtle differences, were used in these definitions. Ullman [10], Van Gelder and Topor [12], and Abiteboul *et al.* [1] gave algorithms for translating queries in these classes into relational algebra for efficient evaluation.

Other researchers such as Escobar-Molano *et al.* [4], Hull and Su [5] and Suciu [8] attempted to define decidable classes of queries, *safe* queries, that were guaranteed to be finite in the presence of functions (e.g., arithmetic functions) over infinite domains (e.g., the natural numbers).

Whether or not there is a decidable class of queries that can express every finite (resp., domain independent) query depends critically on the particular set of functions on the domains. Stolboushkin and Taitslin [7] showed that, for many common domains, there is a decidable class of queries that can express every finite (resp. domain independent) query, but that there do exist domains for which there is no such decidable class. Benedikt and Libkin [2] extended and generalized these results to a wider class of domains.

Foundations

Following standard practice in the literature, assume that every database is defined over a single domain. The use of multiple domains complicates the presentation without introducing any substantially new concepts.

A *domain* $D = (U, O)$ consists of an underlying set U and a set of operations O on the set. The set may be infinite or finite. The operations may be represented as (infinite) relations over the set. Technically, each operation must be decidable. Often, one also wants the first-order theory of the domain to also be decidable. These conditions are satisfied in most common cases.

Examples of such domains include (a) any finite set of symbols possibly with an equality operator, (b) the set of natural numbers with addition and linear order operators, and (c) the set of finite strings over some finite alphabet with concatenation and lexicographic order operators.

A *database scheme* S is a finite set of pairs $\{(S_i, p_i) \mid 1 \leq i \leq k\}$, where each S_i is a relation name with arity $p_i \geq 1$.

Given a domain $D = (U, O)$, a *database (instance)* I of a scheme $\{(S_i, p_i) \mid 1 \leq i \leq k\}$ over D is a family of finite sets $\{R_i \mid 1 \leq i \leq k\}$, where each $R_i \subseteq U^{p_i}$.

A *query* over a database scheme S and domain $D = (U, O)$ is called *finite* if, for every database instance I of S over D , the value of Q when applied to I is a finite relation over U .

A query Q over a database scheme S and domain $D = (U, O)$ is called *finite* if, for every database instance I of S over D , the value of Q when applied to I is a finite relation over U .

For example, over the domain of natural numbers, the query $Q_1 = \{n \mid n + 1 \leq 10\}$ is finite, but the query $Q_2 = \{n \mid 10 \leq n + 1\}$ is infinite.

It is desirable that queries be finite so that they may be composed and so that their results may be displayed.

The *active domain* of a database I is the finite set of domain elements that occur in the relations of I .

A query Q over a database scheme S and domain $D = (U, O)$ is called *domain independent (d.i.)* if, for every database instance I of S over D , the value of Q when applied to I contains only elements in the active domain of I . Equivalently, a query Q is domain independent if and only if, for every database instance I ,

and for every two extensions U_1 and U_2 of the active domain of D , the value of Q when applied to I over U_1 equals the value of Q when applied to I over U_2 .

For example, over a finite domain of symbols without equality, the query $Q_3 = \{x \mid \exists y(P(x) \vee R(y))\}$ is *not* domain independent because, when applied to any database instance over this domain, the value consists all elements x that occur in P if R is empty, and *all* elements of the domain otherwise.

It is desirable that queries be domain independent so that their values are predictable despite the possibly unknown underlying domain.

It is natural to ask about the difference between finiteness and independence.

Clearly, if the domain is finite, all queries are finite, but the query Q_3 above is still not domain independent.

However, if the domain is infinite and the only operation on the domain is equality, then a query is finite if and only if it is domain independent [7].

Further, if the domain is infinite, even if there are operations other than equality, every domain independent query is also finite (as the active domain of every database instance is finite).

More interestingly, if the domain is the set of natural numbers and the only operation on the domain is linear order, then the query

$$Q_4 = \{x \mid \forall y(\Delta(y) \rightarrow x > y) \wedge \forall y(y < x \rightarrow \exists z(\Delta(z) \wedge z \geq y)),$$

where $\Delta(y)$ is true if and only if y is in the active domain of the database, defines the smallest integer greater than all the active domain elements, and is hence finite but not domain independent [7].

Next, it is natural to ask whether it is possible to effectively recognize finite or domain independent queries. The answer is no. By reduction from standard undecidable problems in first-order logic, DiPaola [3] and, independently, Vardi [11] showed that, over any infinite domain, if the database scheme contains at least one relation of arity 2 or more, the classes of finite and domain independent queries are both undecidable. This undecidability result extends to domains such as the natural numbers with addition and linear order operations.

However, Benedikt and Libkin [2] show that finiteness (resp., d.i.) is decidable for Boolean combinations

of conjunctive queries for a large class of domains over the real numbers.

Given that it is not possible to recognize whether or not a given query is finite (resp., d.i.) over a given domain D , two further questions arise naturally. First, is there a *decidable* class of queries over D that can express all finite (resp., d.i.) queries over D ? That is, is there a decidable class C of queries over D such that, for every finite (resp., d.i.) query Q there exists an equivalent query Q' in C ? Second, can a large, simple, decidable class C' of queries be defined such that every query in C' is finite (resp., d.i.)? Historically, the second question was considered first, but they were considered in the order given.

For many domains, there is a decidable class of queries that can express every finite (resp., d.i.) query. Stolboushkin *et al.* [7] say that the finite (resp. d.i.) queries hence have an “effective syntax” for such domains. Examples of domains for which the finite (resp., d.i.) queries have an effective syntax include (a) an infinite domain of symbols in which the only operation is equality, (b) the domain of natural numbers with only the linear order operation, (c) the domain of natural numbers with the addition and linear order operations (Pressburger arithmetic), and (d) the domain of finite strings over a finite alphabet with the lexicographic order operation [7].

However, Stolboushkin and Taitslin [7] show that it is possible to construct an (artificial) domain with decidable operations and decidable first-order theory that does *not* have an effective syntax for the finite (resp., d.i.) queries. They also give an example of a domain with *undecidable* first-order theory (arithmetical) that has an effective syntax for the finite (resp., d.i.) queries.

Given the undecidability results above, many researchers, *e.g.*, [1,6,9,10]) have defined specific, decidable classes of queries that are guaranteed to be finite (resp. d.i.). Researchers have attempted to make these classes both simple and as large as possible. In many cases, they showed that queries in their class could express *all* finite (resp. d.i.) queries. These classes were given names such as safe, range-restricted, allowed, and many others. Most researchers restricted attention to domains with equality as the only operation, but some extended their work to the domain of the natural numbers with arithmetic and linear order operations, and some, *e.g.*, [2,4,5]) considered

more arbitrary domains. Others, *e.g.*, [9]) applied these ideas to the domain independence of deductive databases.

The basic idea of these definitions is to ensure that every free variable in the query is somehow bound to an element in the active domain of the database or, in the presence of nontrivial operations, to one of a finite number of domain elements. In the absence of operations, this is typically done by ensuring that every free or existentially quantified variable in a query occurs positively in its scope, every universally quantified variable occurs negatively in its scope, and that the same free variables occur in every component of a disjunction. For example, the query $\{x \mid P(x) \wedge \forall y(Q(x, y) \rightarrow R(x, y))\}$ is safe according to these ideas. The equality operation is used to propagate a positive variable (or constant) from one side of the equality to the other. For example, the query $\{x \mid P(y) \wedge x = y\}$ is safe according to this idea. Finiteness dependencies are used with arithmetic operations over the natural numbers of concatenation operations over strings to propagate a positive variable (or constant) from one or more positions in the operation to one or more other positions. For example, the query $\{x \mid P(z) \wedge x + y = z \wedge \neg Q(y)\}$ is safe according to this idea. With deductive databases or, equivalently, with Datalog queries, it is necessary to require that every variable in the head of a rule occurs positively in the body of a rule and that the body of a rule is itself safe. In every case, the details of the definitions are too complicated to present here.

Finally, as relational calculus queries are evaluated in real database systems by translation into relational algebra, many researchers have studied techniques for translating safe queries (as defined in the previous two paragraphs) into equivalent relational algebra expressions. (A basic result that many researchers proved is that the class of safe queries is equivalent to the class of relational algebra queries.) These translations typically involve a sequence of transformations into increasingly restricted forms, until the translation into relational algebra is direct. Again, the details of the transformations too complicated to present here. See [1,10,12] for more information.

Key Applications

The concepts of finiteness, domain independence and safety are fundamental to our understanding of

database queries over different domains. Tools that generate queries over specific domains should only generate safe queries. Query processors should check that input queries over specific domains are safe and should report warnings otherwise.

Future Directions

These questions of finiteness, domain independence and safety are well-understood and largely resolved for relational databases. However, additional work extending these ideas and methods to other data models and query languages may still be required.

Cross-references

- ▶ [Complete Query Languages](#)
- ▶ [Conjunctive Query Language](#)
- ▶ [Constraint Query](#)
- ▶ [Query Language](#)
- ▶ [Relational Algebra](#)
- ▶ [Relational Calculus](#)
- ▶ [Relational Model](#)

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases, Chapter 5. Addison-Wesley, Reading, MA, 1995, pp. 70–104.
2. Benedikt M. and Libkin L. Safe constraint queries, SIAM J. Comput., 29:1652–1682, 2000.
3. DiPaola R.A. The recursive unsolvability of the decision problem for the class of definite formulas. J. ACM, 16(2):324–327, 1969.
4. Escobar-Molano M., Hull., and Jacobs D. Safety and translation of calculus queries with scalar functions. In Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1993, pp. 253–264.
5. Hull R. and Su J. Domain independence and the relational calculus. Acta Inform., 31:513–524, 1994.
6. Kifer M. On Safety, Domain Independence, and Capturability of Database Queries (Preliminary Report). In Proc. 3rd Int. Conf. on Data and Knowledge Bases, 1988, pp. 405–415.
7. Stolboushkin A.P. and Taitslin M.A. Finite queries do not have effective syntax, Inform. Comput., 153(1):99–116, 1996.
8. Suciu D. Domain-independent queries on databases with external functions. Theor. Comput. Sci., 190(2):279–315, 1998.
9. Topor R.W. Domain independent formulas and databases. Theor. Comput. Sci., 52(3):281–306, 1987.
10. Ullman J.D. Principles of Database and Knowledge-Base Systems, Volume I, Sections 3.2 and 3.8. Computer Science Press, 1988, pp. 100–106 and 145–156.
11. Vardi M.Y. The decision problem for database dependencies. Inform. Process. Lett., 13(5):251–254, 1981.
12. Van Gelder A. and Topor R.W. Safety and translation of relational calculus queries. ACM Trans. Database Syst., 16(2):235–278, 1981.

Sagas

KENNETH SALEM

University of Waterloo, Waterloo, ON, Canada

Definition

A saga [3] is a sequence of atomic transactions T_1, \dots, T_n for which the following execution guarantee is made. Either the component transactions T_i will all commit in the order

$$T_1 \ T_2, \dots, T_n$$

in which case that saga is said to have committed, or one of the transaction sequences

$$T_1, \dots, T_j \ C_j, \dots, C_1$$

will be executed (for some $0 \leq j < n$), in which case the saga is said to have aborted. The transactions C_i are *compensating transactions* for the corresponding saga transactions T_i . Each transaction T_i in a saga must have a corresponding compensating transaction, which is responsible for undoing the T_i 's effects.

Key Points

A saga is a type of extended transaction model [1]. Each component transaction in a saga is executed atomically, but the saga itself is not atomic. Effects of the component transactions are visible to other operations as soon as those transactions commit, which may be well before the saga has finished.

The saga model guarantees execution of a compensating transaction for each component transaction that has already committed at the time that a saga aborts. This guarantee is known as *semantic atomicity* [2]. The saga model does not specify the nature of these compensations. Rather, the definition or identification of appropriate compensations is an application-specific task.

Sagas were originally proposed as a weaker substitute for the traditional atomic transaction model in situations for which atomicity would be expensive to enforce, e.g., when the traditional transaction would be long-running. Sagas and other extended transaction models have since found a variety of applications, such as workflow systems.

Cross-references

- ▶ [Compensating Transactions](#)
- ▶ [Extended Transaction Model](#)

- ▶ Open Nested Transaction Models
- ▶ Semantic Atomicity
- ▶ Workflow Management

Recommended Reading

1. Chrysanthis P.K. and Ramamritham K. Synthesis of extended transaction models using ACTA. *ACM Trans. Database Syst.*, 19(3):450–491, 1994.
2. Garcia-Molina H. Using semantic knowledge for transaction processing in a distributed database. *ACM Trans. Database Syst.*, 8(2):186–213, 1983.
3. Garcia-Molina H. and Salem K. Sagas. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 249–259.

Samba

- ▶ Storage Protocols

Sampling

- ▶ Matrix Masking

Sampling Techniques for Statistical Databases

AMARNATH GUPTA

University of California-San Diego, La Jolla, CA, USA

Definition

A sampling technique is a method by which one inspects only a small portion of data from a database to reduce the time to compute an aggregate query, but simultaneously ensuring that result computed on the sample faithfully represents the true results of the query for the entire data population.

Example: Acceptance-Rejection sampling (AR sampling) is sampling technique.

Key Points

Sampling is used in a database for different reasons such as (i) to estimate the results of aggregate queries

(e.g., SUM, COUNT, or AVERAGE), (ii) to retrieve a sample of records from a database query for subsequent processing, (iii) for internal use by the query optimizer for selectivity estimation, (iv) to provide privacy protection for records on individuals contained in statistical databases. It has been determined that fixed size random sampling of data does not yield a true representation of the population. *Acceptance/rejection (A/R) sampling* is used to construct weighted samples in which the inclusion probabilities of a record are proportional to some arbitrary weight. *Reservoir sampling* is a form of sequential scan sampling algorithms which are used on files of unknown size to perform on-the-fly sampling from the results of a query. Methods have been developed to perform sampling not only from raw records, but also from B+-trees, hash structures, spatial data structures and so on.

Cross-references

- ▶ On-line Analytical Processing
- ▶ Privacy
- ▶ Secure Database Development
- ▶ Summarizability

Recommended Reading

1. Olken F. and Rotem D. Random sampling from databases: a survey. *Stat. Comput.*, 5:25–42, 1995.

SAN

- ▶ Storage Area Network

SAN File System

KAZUO GODA

The University of Tokyo, Tokyo, Japan

Synonyms

Shared-Disk File System

Definition

The term SAN file system refers to a file system which transfers file data directly to/from a storage device through a SAN. A SAN file system often has the

capability of coherency control such that multiple servers may share the file system volume and simultaneously access files stored in the volume. The term shared disk file system is also used to refer to a SAN file system.

Key Points

In contrast to local file systems, SAN file systems allow multiple servers to share file system volumes directly and to access the same file simultaneously. To achieve this, the SAN file system has the capability of coherency control between servers. File system software running on each server may cache file data in a main memory buffer. Suppose that two servers, A and B, have cached the same file X. If A and B were to update X independently at the same time, two versions of X might be generated. The SAN file system needs to let each server be aware of the other servers. When server A updates a fragment of the file X, server B must be informed of a message of cache invalidation or changed information regarding the file X. Some SAN file systems exchange mutual exclusion messages to synchronize write accesses between servers.

The beneficial property of SAN file systems is high performance in comparison with network file systems such as NFS and CIFS. SAN file systems can directly transfer file data between servers and storage devices e.g., on a high-speed Fibre Channel network. In addition, SAN file systems do not need a central file server, which often becomes a bottleneck.

A variety of SAN file systems have been proposed and some of them have been deployed mainly into cluster computer systems often used to run scientific calculations. Recently, SAN file systems are also used as back-end file systems for large-scale enterprise NAS systems.

Cross-references

► Storage Network Architectures

Recommended Reading

- Barrios M., Jones T., Kinnane S., Landzettel M., Al-Safran S., Stevens J., Stone C., Thomas C., and Troppens U. Sizing and tuning GPFS. IBM Redbook. SG24-5610-00, 1999.
- Burns R.C., Rees R.M., and Long D.D.E. Semi-preemptible locks for a distributed file system. In Proc. 19th IEEE Int. Performance, Computing and Communications Conf., 2000, pp. 397–404.

- Soltis S.R., Ruwart T.M., and O'Keefe M.T. The global file system. In Proc. 15th NASA Goddard Conference on Mass Storage Systems, 1996, pp. 319–342.

SARBAC

► Administration Model for RBAC

SAS

► Storage Protocols

SATA

► Storage Protocols

SBQL

► Stack-Based Query Language

SCA

► Service Component Architecture (SCA)

Scalable Classification Tree Construction

► Scalable Decision Tree Construction

Scalable Database Replication

► Replication for Scalability

Scalable Decision Support Systems High Performance Data Warehousing

► Parallel and Distributed Data Warehouses

Scalable Decision Tree Construction

JOHANNES GEHRKE
Cornell University, Ithaca, NY, USA

Synonyms

Scalable classification tree construction; Scalable top-down decision tree construction; Tree-structured classifier

Definition

Decision trees are popular classification models. Decision trees are usually constructed greedily top-down from a training dataset. In many modern applications, the training dataset is very large and thus decision tree construction algorithms that scale with the size of the training dataset are needed.

Historical Background

Decision trees, in particular classification trees, have a long history both in the statistics [4] and the machine learning communities [12,13]. Scalability was not much a concern until the advent of data mining brought training datasets that were orders of magnitude larger than in traditional applications in machine learning and statistics.

Scalability concerns in classification started with the work by Agrawal et al. who presented an interval classifier that generated classification functions that distinguishes the different groups of training records based on their class label [1]. A follow-up paper introduces scalable construction of classification models as one of the three important classes of database mining problems [2], the other two being associations and sequences. The first scalable classification tree construction algorithm in the literature was SLIQ [9], which was then quickly followed by more algorithms that improved performance and allowed scaling up a more general class of algorithms from the machine learning and statistics literature [5,6,14–17].

Foundations

The input to a classification or regression problem is a dataset of *training records* (also called the *training database*). Each record has several attributes. Attributes whose domain is numerical are called *numerical attributes*, whereas attributes whose domain is not numerical are called *categorical attributes*. A *categorical*

attribute takes values from a set of categories. Some authors distinguish between categorical attributes that take values in an unordered set (*nominal attributes*) and categorical attributes having ordered domains (*ordinal attributes*).

There is one distinguished attribute called the *dependent attribute*. The remaining attributes are called *predictor attributes*; they are either numerical or categorical. If the dependent attribute is categorical, the problem is referred to as a *classification problem* and the dependent attribute is called the *class label*. The elements of the domain of the class label attribute will also be denoted as *class labels*; the meaning of the term class label will be clear from the context. If the dependent attribute is numerical, the problem is called a *regression problem*. This entry concentrates on classification problems.

The goal of classification is to build a concise model of the distribution of the dependent attribute in terms of the predictor attributes. The resulting model is used to assign values to a database where the values of the predictor attributes are known but the value of the dependent attribute is unknown. This entry surveys research on scalable classification tree construction from the database literature. An excellent survey of other aspects of decision tree construction can be found in Murthy [11].

Problem Definition

Let X_1, \dots, X_m, C be random variables where X_i has domain $\text{dom}(X_i)$; assume without loss of generality that $\text{dom}(C) = \{1, 2, \dots, J\}$. A *classifier* is a function

$$d : \text{dom}(X_1) \times \dots \times \text{dom}(X_m) \mapsto \text{dom}(C).$$

Let $P(X', C')$ be a probability distribution on $\text{dom}(X_1) \times \dots \times \text{dom}(X_m) \times \text{dom}(C)$ and let $t = \langle t.X_1, \dots, t.X_m, t.C \rangle$ be a record randomly drawn from P , i.e., t has probability $P(X', C')$ that $\langle t.X_1, \dots, t.X_m \rangle \in X'$ and $t.C \in C'$. Define the *misclassification rate* R_d of classifier d to be $P(d(\langle t.X_1, \dots, t.X_m \rangle) \neq t.C)$. The training database D is a random sample from P ; the X_i correspond to the predictor attributes and C is the class label attribute.

A *decision tree* is a special type of classifier. It is a directed, acyclic graph T in the form of a tree. The focus here is on binary decision trees although these techniques can be generalized to non-binary decision trees. If a node has no outgoing edges it is called a *leaf node*, otherwise it is called an *internal node*. Each leaf

node is labeled with one class label; each internal node n is labeled with one predictor attribute X_n called the *splitting attribute*. Each internal node n has a predicate q_n called the *splitting predicate* associated with it. If X_n is a numerical attribute, q_n is of the form $X_n \leq x_n$, where $x_n \in \text{dom}(X_n)$; x_n is called the *split point* at node n . If X_n is a categorical attribute, q_n is of the form $X_n \in Y_n$ where $Y_n \subset \text{dom}(X_n)$; Y_n is called the *splitting subset* at node n . The combined information of splitting attribute and splitting predicates at node n is called the *splitting criterion* of n . An example training database is shown in Fig. 1, and a sample classification tree is shown in Fig. 2.

With each node $n \in T$ there is associated a predicate

$$f_n : \text{dom}(X_1) \times \dots \times \text{dom}(X_m) \mapsto \{\text{true}, \text{false}\},$$

called its *node predicate* as follows: For the root node n , $f_n \stackrel{\text{def}}{=} \text{true}$. Let n be a non-root node with parent p whose splitting predicate is q_p . If n is the right child of p , define $f_n \stackrel{\text{def}}{=} f_p \wedge q_p$; if n is the left child of p , define $f_n \stackrel{\text{def}}{=} f_p \wedge \neg q_p$. Informally, f_n is the conjunction of all splitting predicates on the internal nodes on the path from the root node to n . Since each leaf node $n \in T$ is labeled with a class label, n encodes the classification rule $f_n \rightarrow c$, where c is the label of n . Thus the tree T encodes a function $T : \text{dom}(X_1) \times \dots \times \text{dom}(X_m) \mapsto \text{dom}(C)$ and is therefore a classifier, called a *decision tree classifier*. (Both the tree as well as the induced classifier will be denoted by T ; the semantics will be clear from the context.) For a node $n \in T$ with parent p , the *family of tuples* F_n is the set of records in D that

follows the path from the root to n when being processed by the tree, formally

$$F_n \stackrel{\text{def}}{=} \{t \in D : f_n(t)\}.$$

Also define F_n^i for $i \in \{1, \dots, J\}$ as the set of records in F_n with class label i , formally

$$F_n^i \stackrel{\text{def}}{=} \{t \in D : f_n(t) \wedge t.C = i\}.$$

The problem of classification tree construction can now be stated formally: Given a dataset $D = \{t_1, \dots, t_n\}$ where the t_i are independent random samples from an unknown probability distribution P , find a decision tree classifier T that minimizes the misclassification rate $R_T(P)$.

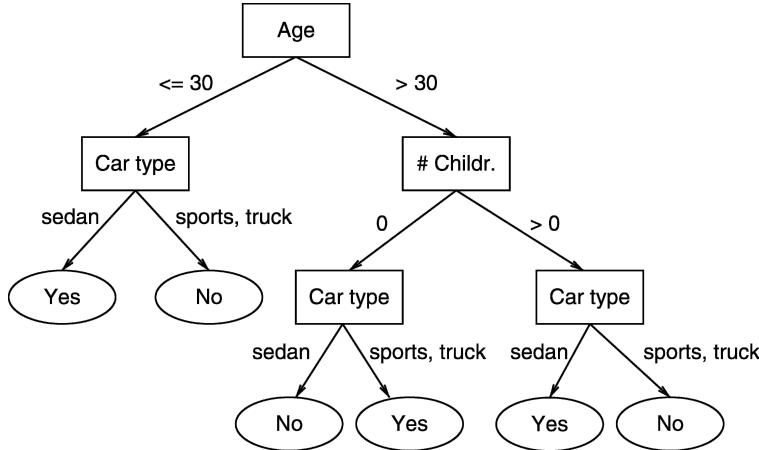
A classification tree is usually constructed in two phases. In phase one, the *growth phase*, an overly large decision tree is constructed from the training data. In phase two, the *pruning phase*, the final size of the tree T is determined with the goal to minimize R_T . It is possible to interleave growth and pruning phase for performance reasons as in the PUBLIC Pruning Method described later in this entry. Nearly all decision tree construction algorithms grow the tree top-down in the following greedy way: At the root node n , the training database is examined and a splitting criterion for n is selected. Recursively, at a non-root node n , the family of n is examined and from it a splitting criterion is selected. This schema is depicted in Fig. 3.

During the tree growth phase, two different algorithmic issues need to be addressed. The first issue is to devise an algorithm such that the resulting tree T minimizes R_T ; this part of the overall decision tree construction algorithm is called the *split selection method*. The second issue is to devise a *data access method* for data management in the case that the training database is very large. During the pruning phase a third issue arises, namely how to find a good estimator \widehat{R}_T of R_T and how to efficiently calculate \widehat{R}_T .

A popular class of split selection methods are *impurity-based* split selection methods [4, 12]. Impurity-based split selection methods find the splitting criterion by minimizing a concave *impurity function* imp_θ such as the entropy [12] or the gini-index [4]. (Arguments for the concavity of the impurity function can be found in Breiman et al. [4].) The most popular split selection methods such as CART [4] and C4.5 [12] fall

Record Id	Car	Age	Children	Subscription
1	sedan	23	0	yes
2	sports	31	1	no
3	sedan	36	1	no
4	truck	25	2	no
5	sports	30	0	no
6	sedan	36	0	no
7	sedan	25	0	yes
8	truck	36	1	no
9	sedan	30	2	yes
10	sedan	31	1	yes
11	sports	25	0	no
12	sedan	45	1	yes
13	sports	23	2	no
14	truck	45	0	yes

Scalable Decision Tree Construction. Figure 1. Example training database.



Scalable Decision Tree Construction. Figure 2. Magazine Subscription Example Classification Tree.

into this group. At each node, all predictor attributes X are examined and the impurity imp_θ of the best split on X is calculated. The final split is chosen such that the combination of splitting attribute and splitting predicates minimizes the value of imp_θ .

Data Access

There exist many scalable data access methods for classification tree construction. Some of the issues in scalable decision tree construction are introduced by briefly discussing one method, RainForest [6].

An examination of the split selection methods in the literature reveals that the greedy schema can be refined to the generic *RainForest Tree Induction Schema* shown in Fig. 4. A broad class of split selection methods, namely those that generate splitting criteria involving a single splitting attribute, proceed according to this generic schema. Split selection methods that generate linear combination splits cannot be captured by RainForest. Consider a node n of the decision tree. The split selection method has to make two decisions while examining the family of n : (i) It has to select the splitting attribute X , and (ii) it has to select the splitting predicates on X . Once decided on the splitting criterion, the algorithm is recursively applied to each of the children of n . Denote by SS a representative split selection method.

Note that at a node n , the utility of a predictor attribute X as a possible splitting attribute is examined independent of the other predictor attributes: The *sufficient statistics* are the class label distributions for each distinct attribute value of X . Define the *AVC-set* of a predictor attribute X at node n to be the projection

Input: Node n , partition D , split selection method SS

Output: Decision tree for D rooted at node n

Top-down decision tree induction schema:

BuildTree (Node n , dataset D , split selection method SS)
(1) Apply SS to D to find the splitting criterion
(2) **if** n splits
(3) Use best split to partition D into D_1 and D_2
(4) BuildTree(n_1, D_1, SS)
(5) BuildTree(n_2, D_2, SS)
(6) **endif**

Scalable Decision Tree Construction. Figure 3.

Classification tree construction.

of F_n onto X and the class label where counts of the individual class labels are aggregated. Denote the AVC-set of predictor attribute X at node n by $\text{AVC}_n(X)$. (The acronym AVC stands for Attribute-Value, Classlabel.) To give a formal definition, let $a_{n,X,x,i}$ be the number of records t in F_n with attribute value $t.X = x$ and class label $t.C = i$. Formally,

$$a_{n,X,x,i} \stackrel{\text{def}}{=} |\{t \in F_n : t.X = x \wedge t.C = i\}|.$$

For a predictor attribute X , let $S \stackrel{\text{def}}{=} \text{dom}(X) \times \mathbf{N}^I$ where \mathbf{N} denotes the set of natural numbers. Then

$$\text{AVC}_n(X) \stackrel{\text{def}}{=} \{(x, a_1, \dots, a_J) \in S : \exists t \in F_n : (t.X = x \wedge \forall i \in \{1, \dots, J\} : a_i = a_{n,X,x,i})\}.$$

Define the *AVC-group* of a node n to be the set of the AVC-sets of all predictor attributes at node n . Note that the size of the AVC-set of a predictor attribute X at

RainForest refinement to the schema in Fig 3:

- (1a) **for** each predictor attribute X
- (1b) Construct the AVC-set of X
- (1c) Call $SS.\text{find_best_partitioning}(\text{AVC-set of } X)$
- (1d) **endfor**
- (2a) $SS.\text{decide_splitting_criterion}();$
- (2b) **if** n splits...

Scalable Decision Tree Construction. Figure 4.

RainForest refinement.

Car	Subscription	
	Yes	No
sedan	5	2
sports	0	4
truck	1	2

Age	Subscription	
	Yes	No
23	1	1
25	1	2
30	1	1
31	1	1
36	0	3
45	2	0

Scalable Decision Tree Construction. Figure 5.

Rainforest AVC-sets.

node n depends only on the number of distinct attribute values of X and the number of class labels in F_n .

As an example, consider the training database shown in Fig. 1. The AVC group of the root node is depicted in Fig. 5. Assume that the root node splits as shown in Fig. 2. The AVC-group of the left child node of the root node is shown in Fig. 5 and the AVC-group of the left child node of the root node is shown in Fig. 6.

If the training database is stored inside a database system, the AVC-set of a node n for predictor attribute X can be retrieved through a simple SQL-query:

```
SELECT      D.X, D.C, COUNT(*)
FROM        D
WHERE       fn
GROUP BY    D, X, D.C
```

In order to construct the AVC-sets of all predictor attributes at a node n , a UNION-query would be necessary. (In this case, the SELECT clause needs to retrieve also some identifier of the attribute in order to distinguish individual AVC-sets.) Graefe et al. observe that most database systems evaluate the UNION-query through several scans and introduce a new operator that allows gathering of sufficient statistics in one database scan [7].

Based on this observation, there exist several algorithms that construct as many AVC-sets as possible in main memory while minimizing the number of scans over the training database. As an example of the simplest such algorithm, assume that the complete AVC-group of the root node fits into main memory. Then the tree can be constructed according to the following simple schema: Read the training database D and construct the AVC-group of the root node n in-memory. Then determine the splitting criterion from the AVC-sets through an in-memory computation. Then make a second pass over D and partition D into children

Car	Subscription	
	Yes	No
sedan	3	0
sports	0	3
truck	0	1

Age	Subscription	
	Yes	No
23	1	1
25	1	2
30	1	1

Scalable Decision Tree Construction. Figure 6.

Rainforest AVC-sets of the left child of the root node.

partitions D_1 and D_2 . This simple algorithm reads the complete training database twice and writes the training database once per level of the tree; more sophisticated algorithms are possible [6]. Experiments show that RainForest outperforms SPRINT on the average by a factor of three. Note that RainForest has a large memory requirement: RainForest is only applicable if the AVC-group of the root node fits in-memory (this requirement can be relaxed through more sophisticated memory management [6]).

Tree Pruning

The pruning phase of classification tree construction decides on the tree of the right size in order to prevent overfitting to minimize the misclassification error $R_T(P)$. In bottom-up pruning, in the tree growth phase the tree is grown until the size of the family of each leaf node n falls below a user-defined threshold c ; the pruning phase follows the growth phase. Examples of bottom-up pruning strategies are cost-complexity pruning, pruning with an additional set of records called a test set [4], and pruning based on the MDL-principle [10]. In top-down pruning, during the growth phase a statistic s_n is computed at each node n , and based on the value of s_n , tree growth at node n is continued or stopped [13]. Bottom-up pruning results usually in trees of higher quality [4,8,13],

but top-down pruning is computationally more efficient since no parts of the tree are first constructed and later discarded.

The section describes the PUBLIC pruning algorithm [14], an algorithm that integrates bottom-up pruning into the tree growth phase; thus PUBLIC preserves the computational advantages of top-down pruning while preserving the good properties of top-down pruning. PUBLIC uses pruning based on the MDL principle [10], which sees the classification tree as a means to encode the values of the class label attribute given the predictor attributes X_1, \dots, X_m . The MDL principle states that the “best” classification tree is the tree can be encoded with the least number of bits. Thus there needs to be an encoding schema that allows encoding of any binary decision tree. Given an encoding schema, a classification tree can be pruned by selecting the subtree with minimum code length.

In the MDL encoding schema for binary splitting predicates from Mehta et al. [10], each node requires one bit to encode its type (leaf or intermediate node). An intermediate node n needs to encode its splitting criterion, consisting of the splitting attribute X ($\log m$ bits since there are m predictor attributes) and splitting predicate. Let X be the splitting attribute at node n and assume that X has v different attribute values. If X is a numerical attribute, the split will be of the form $X \leq c$. Since c can take $v - 1$ different values, encoding of the split point c requires $\log(v - 1)$ bits. If X is a categorical attribute, the split will be of the form $X \in Y$. Since Y can take $2^v - 2$ different values, the encoding requires $\log(2^v - 2)$ bits. Denote the cost of a split at a node n by $C_{\text{Split}}(n)$. For a leaf node n , Metha et al. show that the cost of encoding the leaf is [10]:

$$C_{\text{leaf}}(n) = \sum_i n_i \log \frac{|F_n|}{|F_n^i|} + \frac{k-1}{2} \log \frac{|F_n|}{2} + \log \frac{\pi^{k/2}}{\Gamma(k/2)}.$$

Given this encoding schema, a fully grown tree can be pruned bottom-up by deciding for each node whether it should be pruned or whether it should remain [10].

The PUBLIC algorithm integrates the building and pruning phase by computing a lower bound $L(n)$ on the MDL-cost of any subtree rooted at a node n . A trivial lower bound is $L(n) = 1$ (for the encoding of n). Rastogi and Shim give in their PUBLIC algorithms more sophisticated lower bounds including the following:

PUBLIC Lower Bound [14]

Consider a (sub-)tree T with $s > 1$ nodes rooted at node n . Then the cost $C(n)$ of encoding T has the following lower bound:

$$C(n) \geq 2 \cdot (s-1) + 1 + (s-1) \cdot \log m + \sum_{i=s+1}^k |F_n^i|$$

With this lower bound the MDL Pruning Schema can be used even during top-down tree construction. PUBLIC distinguishes two different types of leaf nodes: “True” leaf nodes that are the result of pruning or that cannot be expanded any further, and “intermediate” leaf nodes n , where the subtree rooted at n might be grown further. During the growth phase, the PUBLIC Pruning Schema is executed from the root node of the tree. Rastogi and Shim show experimentally that integration of pruning with the growth phase of the tree results in significant savings in overall tree construction. More sophisticated ways of integrating tree growth with pruning in addition to tighter lower bounds are possible [14].

Key Applications

Classification has a wide range of applications, including scientific experiments, medical diagnosis, fraud detection, credit approval, and target marketing.

Cross-references

- ▶ Classification
- ▶ Data Mining

Recommended Reading

1. Agrawal R., Ghosh S.P., Imielinski T., Iyer B.R., and Swami A.N. An interval classifier for database mining applications. In Proc. 18th Int. Conf. on Very Large Data Bases, 1992. pp. 560–573.
2. Agrawal R., Imielinski T., and Swami A.N. Database mining: a performance perspective. IEEE Trans. Knowl. Data Eng., 5(6): 914–925, 1993.
3. Alsabti K., Ranka S., and Singh V. Clouds: a decision tree classifier for large datasets. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 2–8.
4. Breiman L., Friedman J.H., Olshen R.A., and Stone C.J. Classification and regression trees. Wadsworth, Belmont, 1984.
5. Gehrke J., Ganti V., Ramakrishnan R., and Loh W.-Y. BOAT – Optimistic decision tree construction. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 169–180.
6. Gehrke J., Ramakrishnan R., and Ganti V. Rainforest – a framework for fast decision tree construction of large datasets. Data Min. Knowl. Dis., 4(2/3):127–162, 2000.
7. Graefe G., Fayyad U., and Chaudhuri S. On the efficient gathering of sufficient statistics for classification from large

- SQL databases. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998, pp. 204–208.
8. Lim T.-S., Loh W.-Y., and Shih Y.-S. A comparison of prediction accuracy, complexity, and training time of 33 old and new classification algorithms. *Mach. Learn.*, 48:203–228, 2000.
 9. Mehta M., Agrawal R., and Rissanen J. SLIQ: A fast scalable classifier for data mining. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996.
 10. Mehta M., Rissanen J., and Agrawal R. MDL-based decision tree pruning. In Proc. 1st Int. Conf. on Knowledge Discovery and Data Mining, 1995.
 11. Murthy S.K. Automatic construction of decision trees from data: a multi-disciplinary survey. *Data Min. Knowl. Dis.*, 2(4): 345–389, 1998.
 12. Quinlan J.R. Induction of decision trees. *Mach. Learn.*, 1:81–106, 1986.
 13. Quinlan J.R. C4.5: Programs for Machine Learning. Morgan Kaufman, 1993.
 14. Rastogi R. and Shim K. PUBLIC: a decision tree classifier that integrates building and pruning. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 404–415.
 15. Shafer J., Agrawal R., and Mehta M. SPRINT: a scalable parallel classifier for data mining. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996.
 16. Sreenivas M.K., AlSabti K., and Ranka S. Parallel out-of-core decision tree classifiers. In Advances in Distributed and Parallel Knowledge Discovery. Kargupta H. and Chan P. (eds.). AAAI. 2000, pp. 317–336.
 17. Srivastava A., Han E., Kumar V., and Singh V. Parallel formulations of decision-tree classification algorithms. *Data Min. Knowl. Dis.*, 3(3), 1999.

Scalable Replication

- Replication in Multi-Tier Architectures

Scalable Top-Down Decision Tree Construction

- Scalable Decision Tree Construction

Scale Out

- Replication for Scalability
- Replication in Multi-Tier Architectures

Scale-Out Databases

- Process Structure of a DBMS

Scale-Up Databases

- Process Structure of a DBMS

Scaling

- Zooming Techniques

Scanning

- Browsing

Scene Change Detection

- Video Segmentation

Scheduler

NATHANIEL PALMER

Workflow Management Coalition, Hingham,
MA, USA

Synonyms

Workflow scheduler; Queuing mechanism

Definition

The mechanism that identifies and initiates the sequence for activities and work items are executed.

Key Points

A Scheduler initiates work assignments based on precedence relationships and the state of a workflow instance. A Scheduler is not a “queue” which is typically a set of work items or activities waiting to be scheduled, however, a queue of items may be managed by the Scheduler. The role of the Scheduler is to minimize queue time and optimize executive efficiency.

Cross-references

- Activity
- Work Item

Scheduling

- Concurrency Control – Traditional Approaches

Scheduling Policies

- Scheduling Strategies for Data Stream Processing

Scheduling Strategies for Data Stream Processing

MOHAMED SHARAF¹, ALEXANDROS LABRINIDIS²

¹Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada

²Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

Operator scheduling; Continuous query scheduling;
Scheduling policies

Definition

In a Data Stream Management System (DSMS), data arrives in the form of continuous streams from different data sources, where the arrival of new data triggers the execution of multiple continuous queries (CQs). The order in which CQs are executed in response to the arrival of new data is determined by the CQ scheduler. Thus, one of the main goals in the design of a DSMS is the development of scheduling policies that leverage CQ characteristics to optimize the DSMS performance.

Historical Background

The growing need for *monitoring applications* [8] has forced an evolution on data processing paradigms, moving from Database Management Systems (DBMSs) to Data Stream Management Systems (DSMSs) [4,11]. Traditional DBMSs employ a store-and-then-query data processing paradigm, where data are stored in the database and queries are submitted by the users to be answered in full, based on the current snapshot of the database. In contrast, in DSMSs, monitoring applications register continuous queries which continuously process unbounded data streams looking for data that represent events of interest to the end-user.

The data stream concept permeated the data management research community in the mid- to late 90's,

with general-purpose research prototypes of data stream management systems materializing shortly afterwards, for example Aurora[8], TelegraphCQ[10] and STREAMS[5].

Scheduling is one of the fundamental research challenges for effective data stream management systems; as such, it has received a lot of attention, with early works on scheduling in 2003 [2,9].

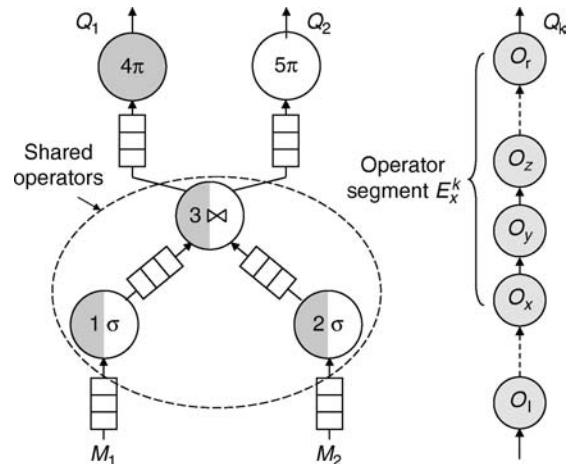
Foundations

System Model

A continuous query evaluation plan can be conceptualized as a data flow tree [2,8], where the nodes are operators that process tuples and edges represent the flow of tuples from one operator to another (Fig. 1). An edge from operator O_x to operator O_y means that the output of O_x is an input to O_y . Each operator is associated with a *queue* where input tuples are buffered until they are processed.

Multiple queries with common sub-expressions are usually merged together to eliminate the repetition of similar operations. For example, Figure 1 shows the global plan for two queries Q_1 and Q_2 . Both queries operate on data streams M_1 and M_2 and they share the common sub-expression represented by operators O_1 , O_2 and O_3 , as illustrated by the half-shaded pattern for these operators.

A *single-stream query* Q_k has a single *leaf* operator Q_l^k and a single *root* operator Q_r^k , whereas a *multi-stream query* has a single root operator and more than one leaf operators. In a query plan Q_b , an



Scheduling Strategies for Data Stream Processing.

Figure 1. Continuous queries plans.

operator segment $E_{x,y}^k$ is the sequence of operators that starts at O_x^k and ends at O_y^k . If the last operator on $E_{x,y}^k$ is the root operator, then that operator segment is simply denoted as E_x^k . For example, in Fig. 1, $E_1^1 = \langle O_1, O_3, O_4 \rangle$, whereas $E_1^2 = \langle O_1, O_3, O_5 \rangle$.

In a query, each operator O_x^k (or simply O_x) is associated with two parameters:

1. *Processing cost or Processing time* (c_x) is the amount of time needed to process an input tuple.
2. *Selectivity or Productivity* (s_x) is the number of tuples produced after processing one tuple for c_x time units. s_x is less than or equal to 1 for a filter operator and it could be greater than 1 for a join operator.

Multiple CQ Scheduling

At the arrival of new data, the MCQ scheduler decides the execution order of CQs, or more precisely, the execution order of operators within CQs. The execution order is decided with the objective of optimizing the DSMS performance under certain metrics. Towards this, the scheduler assigns a priority to each operator and operators are executed according to these priorities.

For a single-stream query Q_k which consists of operators $\langle O_l^k, \dots, O_x^k, O_y^k, \dots, O_r^k \rangle$ (Fig. 1), the function for computing the priority of operator O_x^k typically involves one or more of the following parameters:

- *Operator Global Selectivity* (S_x^k) is the number of tuples produced at the root O_r^k after processing one tuple along operator segment E_x^k .

$$S_x^k = s_x^k \times s_y^k \times \dots \times s_r^k$$

- *Operator Global Average Cost* (\bar{C}_x^k) is the expected time required to process a tuple along an operator segment C_x^k .

$$\bar{C}_x^k = (c_x^k) + (c_y^k \times s_x^k) + \dots + (c_r^k \times s_{r-1}^k \times \dots \times s_x^k)$$

If O_x^k is a leaf operator ($x = l$), when a processed tuple actually satisfies all the filters in E_l^k , then \bar{C}_l^k represents the ideal total processing cost or time incurred by any tuple produced or emitted by query Q_k . In this case, \bar{C}_l^k is denoted as T_k :

- *Tuple Processing Time* (T_k) is the ideal total processing cost required to produce a tuple by query Q_k .

$$T_k = c_l^k + \dots + c_x^k + c_y^k + \dots + c_r^k$$

The exact priority function depends on the performance metric to optimize, and in turn on the employed scheduling strategy.

Metrics and Strategies

Response Time: Processing a tuple by a CQ might lead to discarding it (if it does not satisfy some filter predicate) or it might lead to producing one or more tuples at the output, which means that the input tuple represents an event of interest to the user who registered the CQ. Clearly, in DSMSs, it is more appropriate to define response time from a data/event perspective rather than from a query perspective as in traditional DBMSs. Hence, the *tuple response time* or *tuple latency* is defined as follows:

Definition 1

Tuple response time, R_i , for tuple t_i is $R_i = D_i - A_i$, where A_i is t_i 's arrival time and D_i is t_i 's output time. Accordingly, the average response time for N tuples is:

$$\frac{1}{N} \sum_{i=1}^N R_i.$$

For a single CQ over multiple data streams, the *Rate-based policy* (*RB*) has been shown to improve the average response time of tuples processed by that CQ [17].

For multiple CQs, the Aurora DSMS [9], uses a two-level scheduling strategy where *Round Robin* (*RR*) is used to schedule queries and *RB* is used to schedule operators within the query. The work in [14] proposes the *Highest Rate* policy (*HR*) which extends the *RB* to schedule both queries and operators. Basically, *HR* views the network of multiple queries as a set of operators and at each scheduling point it selects for execution the operator with the highest priority (i.e., output rate).

Specifically, under *HR*, each operator O_x^k is assigned a value called *global output rate* (GR_x^k). The output rate of an operator is basically the expected number of tuples produced per time unit due to processing one tuple by the operators along the operator segment starting at O_x^k all the way to the root O_r^k . Formally, the output rate of operator O_x^k is defined as follows:

$$GR_x^k = \frac{S_x^k}{\bar{C}_x^k} \quad (1)$$

where S_x^k and \bar{C}_x^k are the operator's global selectivity and global average cost as defined above. The intuition underlying *HR* is to give higher priority to operator paths that are both productive and inexpensive. In other

words, the highest priority is given to the operator paths with the minimum latency for producing one tuple.

Slowdown: Under a heterogeneous workload, the processing requirements for different tuples may vary significantly and average response time is not an appropriate metric, since it cannot relate the time spent by a tuple in the system to its processing requirements. Given this realization, other on-line systems with heterogeneous workloads such as DBMSs, OSs, and Web servers have adopted *average slowdown* or *stretch* [13] as another metric. This motivated considering the stretch metric in [14].

The definition of slowdown was initiated by the database community in [12] for measuring the performance of a DBMS executing multi-class workloads. Formally, the slowdown of a job is the ratio between the time a job spends in the system to its processing demands [13]. In a DSMS, the slowdown of a tuple is defined as follows [14]:

Definition 2

The slowdown, H_i , for tuple t_i produced by query Q_k is $H_i = \frac{R_i}{T_k}$, where R_i is t_i 's response time and T_k is its ideal processing time. Accordingly, the average slowdown for N tuples is: $\frac{1}{N} \sum_{i=1}^N H_i$.

Intuitively, in a general purpose DSMS where all events are of equal importance, a simple event (i.e., an event detected by a low-cost CQ) should be detected faster than a complex event (i.e., an event detected by a high-cost CQ) since the latter contributes more to the load on the DSMS.

The *HR* policy schedules jobs in descending order of output rate which might result in a high average slowdown because a low-cost query can be assigned a low priority since it is not productive enough. Those few tuples produced by this query will all experience a high slowdown, with a corresponding increase in the average slowdown of the DSMS.

The work in [14] proposes the *Highest Normalized Rate (HNR)* policy for minimizing the slowdown in a DSMS. Under HNR, each operator O_x^k is assigned a priority V_x^k which is the *weighted rate* or *normalized rate* of the operator segment E_x^k that starts at operator O_x^k and it is defined as:

$$V_x^k = \frac{1}{T_k} \times \frac{S_x^k}{C_x^k} \quad (2)$$

The *HNR* policy, like *HR*, is based on output rate, however, it also emphasizes the ideal tuple processing time in assigning priorities. As such, an inexpensive operator segment with low productivity will get a higher priority under *HNR* than under *HR*.

Worst-Case Performance: It is expected that a scheduling policy that strives to minimize the average-case performance might lead to a poor worst-case performance under a relatively high load. That is, some queries (or tuples) might starve under such a policy. The worst-case performance is typically measured using *maximum response time* or *maximum slowdown* [7].

Intuitively, a policy that optimizes for the worst-case performance should be pessimistic. That is, it assumes the worst-case scenario where each processed tuple will satisfy all the filters in the corresponding query.

The work in [14] shows that the traditional *First-Come-First-Serve (FCFS)* minimizes the maximum response time. Similarly, it shows that the traditional *Longest Stretch First (LSF)* [1] optimizes the maximum slowdown.

Average- vs. Worst-Case Performance: On one hand, the average value for a QoS metric provided by the system represents the expected QoS experienced by any tuple in the system (i.e., the average-case performance). On the other hand, the maximum value measures the worst QoS experienced by some tuple in the system (i.e., the worst-case performance). It is known that each of these metrics by itself is not enough to fully characterize system performance.

The most common way to capture the trade-off between the average-case and the worst-case performance is to measure the ℓ_2 norm [6]. For instance, the ℓ_2 norm of response times, R_p , is defined as:

Definition 3

The ℓ_2 norm of response times for N tuples is equal to $\sqrt{\sum_{i=1}^N R_i^2}$.

The definition shows that the ℓ_2 norm considers the average in the sense that it takes into account all values, yet, by considering the second norm of each value instead of the first norm, it penalizes more severely outliers compared to the average metrics.

In order to balance the trade-off between the average- and worst-case performance, the *Balance Slowdown (BSD)* and the *Balance Response Time (BRT)*

policies have been proposed in [14]. To avoid starvation, the two policies consider the amount of time an operator O_x^k has been waiting for scheduling (i.e., W_x^k). Specifically, under *BSD*, each operator O_x^k is assigned a priority value V_x^k which is the product of the operator's normalized rate and the current highest slowdown of its pending tuples. That is:

$$V_x^k = \left(\frac{S_x^k}{\bar{C}_x T_k} \right) \left(\frac{W_x^k}{T_k} \right) \quad (3)$$

As such, under *BSD*, an operator is selected either because it has a high weighted rate or because its pending tuples have acquired a high slowdown.

Application-Specific QoS: Aurora also proposes a QoS-aware scheduler which attempts to satisfy application-specified QoS requirements [9]. Specifically, under that QoS-aware scheduler, each query is associated with a QoS graph which defines the utility of stale output.

Given, a QoS graph, the scheduler computes for each operator a *utility* value which is basically the slope of the QoS graph at the tuple's output time. The scheduler also computes for each operator its *urgency* value which is an estimation of how close is an operator to a critical point on the QoS graph where the QoS changes sharply. Then, at each scheduling point, the scheduler chooses for execution the operators with the highest utility value and among those that have the same utility, it chooses the one that has the highest urgency.

Memory Usage: Multi-query scheduling has also been exploited to optimize metrics beyond QoS. For example, *Chain* is a multi-query scheduling policy that optimizes memory usage in order to minimize space requirements for buffering tuples [2]. Towards this, for each query plan, *Chain* constructs what is called a *progress chart*. A progress chart is basically a set of segments where the slope of each segment represents the rate of change in the size of a tuple being processed by a set of consecutive operators along the query plan. Given that progress chart, at each scheduling point, *Chain* schedules for execution the tuple that lies on the segment with the steepest slope. The intuition is to give higher priority to segments of operators with higher tuple consumption rate which will lead to quickly freeing more memory.

Quality of Data (QoD): Another metric to optimize is Quality of Data (QoD). For instance, the work in [15] proposes the *freshness-aware* scheduling policy for

improving the QoD of data streams, when QoD is defined in terms of freshness. The proposed scheduler exploits the variability in query costs, divergence in arrival patterns, and the probabilistic impact of selectivity in order to maximize the freshness of output data streams.

Multiple-Objective Scheduling: In DSMSs, and in computer systems in general, it is often desirable to optimize for multiple metrics at the same time. However, those metrics might be in conflict most of the time. This motivated the proposals of schedulers that are able to balance the trade-off between certain conflicting metrics.

For instance, the work in [3] attempts to balance the trade-off between memory usage and latency by formalizing latency requirements as a constraint to the *Chain* scheduler. This formulation lead to the *Mixed* policy which can be viewed as a heuristic strategy that is intermediate between *Chain* and *FIFO*. Specifically, *Mixed* is tuned via a parameter where a high value of that parameter causes *Mixed* to behave more like *FIFO*, whereas a lower value makes it behave more like *Chain*.

In another attempt towards multiple-objective scheduling, the work in [16] proposes *AMoS* which is an Adaptive Multi-objective Scheduling selection framework. Given several scheduling algorithms, *AMoS* employs a learning mechanism to learn the behavior of the scheduling algorithms over time. It then uses the learned knowledge to continuously select the algorithm that has statistically performed the best.

Scheduler Implementation: To ensure the applicability of scheduling policies in DSMSs, a low-overhead implementation is needed in order to reduce the amount of computation involved in computing priorities. For static policies (i.e., policies where an operator priority is constant over time), priorities are computed only once when a query is registered in the DSMS which naturally leads to a low-overhead implementation. Examples of such static policies include *HR*, *HNR*, and *Chain*. On the other hand, for dynamic policies where priority is a function of time, the priority of each operator should be re-computed at each instant of time. Such a naive implementation renders that class of policies very impractical. This motivated several approximation methods for efficient implementation of dynamic policies to balance the trade-off between scheduling overhead and accuracy. For instance the work in [9] proposes using bucketing as well as pre-computation for an efficient

implementation of the QoS-aware scheduling in Aurora. Similarly, [14] proposes using search space reduction and pruning methods in addition to clustered processing of continuous queries.

Key Applications

There is a plethora of applications that require data stream management systems and, as such, proper scheduling strategies. The most well-known class of applications is that of *monitoring applications*[8], be it environmental monitoring (e.g., via sensor networks), network monitoring (e.g., by collecting router data), or even financial monitoring (e.g., by observing stock-market data). In all such cases, the sheer amount of input data precipitates the use of the data stream processing paradigm and proper scheduling strategies.

Cross-references

- ▶ [Adaptive Query Processing](#)
- ▶ [Adaptive Stream Processing](#)
- ▶ [Data Stream](#)
- ▶ [Event Stream](#)
- ▶ [Stream Processing](#)
- ▶ [Stream-Oriented Query Languages and Operators](#)
- ▶ [Streaming Applications](#)

Recommended Reading

1. Acharya S. and Muthukrishnan S. Scheduling on-demand broadcasts: New metrics and algorithms. In Proc. 4th Annual Int. Conf. on Mobile Computing and Networking, 1998.
2. Babcock B., Babu S., Datar M., and Motwani R. Chain: operator scheduling for memory minimization in data stream systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
3. Babcock B., Babu S., Datar M., Motwani R., and Thomas D. Operator scheduling in data stream systems. VLDB J., 13(4), 2004.
4. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and Issues in Data Stream Systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002.
5. Babu S. and Widom J. Continuous queries over data streams. ACM SIGMOD Rec., 2001.
6. Bansal N. and Pruhs K. Server scheduling in the L_p norm: a rising tide lifts all boats. In Proc. 35th Annual ACM Symp. on Theory of Computing, 2003.
7. Bender M.A., Chakrabarti S., and Muthukrishnan S. Flow and stretch metrics for scheduling continuous job streams. In Proc. 9th Annual ACM -SIAM Symp. on Discrete Algorithms, 1998.
8. Carney D., Cetintemel U., Cherniack M., Convey C., Lee S., Seidman G., Stonebraker M., Tatbul N., and Zdonik S. Monitoring streams: a new class of data management applications. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.

9. Carney D., Cetintemel U., Rasin A., Zdonik S., Cherniack M., and Stonebraker M. Operator scheduling in a data stream manager. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003.
10. Chandrasekaran S., Cooper O., Deshpande A., Franklin M.J., Hellerstein J.M., Hong W., Krishnamurthy S., Madden S., Raman V., Reiss F., and Shah M.A. TelegraphCQ: continuous dataflow processing for an uncertain world. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
11. Golab L. and Özsu M.T. Issues in data stream management. ACM SIGMOD Rec., 32(2):5–14, 2003.
12. Mehta M. and DeWitt D.J. Dynamic memory allocation for multiple-query workloads. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993.
13. Muthukrishnan S., Rajaraman R., Shaheen A., and Gehrke J.E. Online Scheduling to Minimize Average Stretch. In Proc. 40th Annual Symp. on Foundations of Computer Science, 1999.
14. Sharaf M.A., Chrysanthis P.K., Labrinidis A., and Pruhs K. Efficient Scheduling of Heterogeneous Continuous Queries. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006.
15. Sharaf M.A., Labrinidis A., Chrysanthis P.K., and Pruhs K. Freshness-Aware Scheduling of Continuous Queries in the Dynamic Web. In Proc. 8th Int. Workshop on the World Wide Web and Database, 2005.
16. Sutherland T., Pielech B., Zhu Y., Ding L., and Rundensteiner E. A. An adaptive multi-objective scheduling selection framework for continuous query processing. In Proc. Int. Database Engineering and Applications Symp, 2005.
17. Urhan T. and Franklin M.J. Dynamic pipeline scheduling for Improving Interactive Query Performance. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.

Schema Evolution

JOHN F. RODDICK

Flinders University, Adelaide, SA, Australia

Definition

Schema evolution deals with the need to retain current data when database schema changes are performed. Formally, *Schema Evolution* is accommodated when a database system facilitates database schema modification without the loss of existing data, (q.v. the stronger concept of *Schema Versioning*) (Schema evolution and schema versioning has been conflated in the literature with the two terms occasionally being used interchangeably. Readers are thus also encouraged to read also the entry for *Schema Versioning*).).

Historical Background

Since schemata change and/or multiple schemata are often required, there is a need to ensure that extant data either stays consistent with the revised schema or

is explicitly deleted as part of the change process. A database that supports schema evolution supports this transformation process.

The first schema evolution proposals discussed database conversion primarily in terms of a set of transformations from one schema to another [10]. These transformations focused on the relational structure of the database and included name changing, changing the membership of keys, composing and decomposing relations both vertically and horizontally and so on. In all cases only one schema remained and all data (that still remained) was coerced (ie. copied from one type to another) to the new structure.

Schema evolution has also been covered in the proposals to manage issues such as data coercion [5,12], authority control [2] and query language support [9].

Foundations

Schema evolution is related to the view-update problem, discussed in-depth when the relational model was introduced [1], and is strongly linked to the notion of information capacity [4,7]. Specifically, non-loss evolution can only be guaranteed when the information capacity of the new schema exceeds that of the existing schema. Formally, if $I(S)$ is the set of all valid instances of S , then for non-loss evolution $I(S_{new}) \supseteq I(S_{old})$. One novel solution is the integration of schema evolution with the database view facilities. When new requirements demand schema updates for a particular user, then the user specifies schema changes to a personal view, rather than to the shared base schema [8].

Once a schema change is accepted, the common procedure is for the underlying instances to be coerced to the new structure. Since the old schema is obsolete, this presents few problems and is conceptually simple. However, results in an inability to reverse schema amendments. Schema versioning support provides two other options (q.v.).

Four classes of schema evolution can be envisaged. Each type brings different problems.

1. *Attribute Evolution* occurs when attributes are added to, deleted from, or renamed in a relation. Issues here include the values to be ascribed to attributes in tuples stored under a new version that does not possess the attribute.
2. *Domain Evolution* occurs when the domain over which an attribute is defined is altered. Issues here include implying accuracy that does not exist in

existing data when, for example, attributes defined as integers are converted to reals, and in truncation when character fields are shortened.

3. *Relation Evolution* occurs when the relational structure is altered through the definition, deletion, decomposition or merging of a relation. Such changes are almost always irreversible.
4. *Key Evolution* occurs when the structure of a primary key is altered or when foreign keys are added or removed. The issues here can be quite complex. For example, removing an attribute from a primary key may not violate the primary key uniqueness constraint for current data (the amendment can be rejected if it does) but in a temporal database may still do so for historical information.

Note that one change may involve more than one type of evolution, such as changing the domain of a key attribute.

These changes may also be reflected in the conceptual model of the system. For example, the addition of an entity in an EER diagram would result in the addition of a relation in the underlying relational model; deleting a 1-to-many relationship would remove a foreign key constraint, and so on.

Key Applications

Schema changes are linked to either error correction or design change. It is therefore useful if the design decisions can be consulted and the users can interact with schema changes at a high level. One way is to propagate requirements changes to database schemas [3] or provide better support for metadata management by providing a higher level view in which models can be mapped to each other [6].

In order to quantify the types of schema evolution, Sjøberg [11] investigated change to a database system over 18 months, covering 6 months of development and 12 months of field trials. A more recent study complements this by following the changes in an established database system over many years [13].

Future Directions

The major directions for schema versioning research have moved from low-level handling of syntactic elemental changes (such as adding an attribute or demoting an index attribute) to more model-directed semantic handling of change (such as propagating changes in a conceptual model to a database schema) [3]. Research

has also moved from schema evolution to the more complex problem of providing versions of schema.

Cross-references

- ▶ Conceptual Modeling
- ▶ Schema Versioning
- ▶ Temporal Algebras
- ▶ Temporal Query Languages

Recommended Reading

1. Bancilhon F. and Spryatos N. Update semantics of relational views. *ACM Trans. Database Syst.*, 6(4):557–575, 1981.
2. Bretl R., Maier D., Otis A., Penney J., Schuchardt B., Stein J., Williams E.H., and Williams M. The GemStone data management system. In *Object-Oriented Concepts, Databases and Applications*. W. Kim and F. Lochovsky (eds.). ACM, New York, NY, USA, 1989, pp. 283–308.
3. Hick J.M. and Hainaut J.L. Database application evolution: a transformational approach. *Data Knowl. Eng.*, 59(3): 534–558, 2006.
4. Hull R. Relative information capacity of simple relational database schemata. *Soc. Ind. Appl. Math.*, 15(3):856–886, 1986.
5. Kim W. and Chou H.T. Versions of schema for object-oriented databases. In Proc. 24th Int. Conf. on Very Large Data Bases, 1988, pp. 148–159.
6. Melnik S., Rahm E., and Bernstein P.A. Rondo: a programming platform for generic model management. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 193–204.
7. Miller R., Ioannidis Y., and Ramakrishnan R. The use of information capacity in schema integration and translation. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 120–133.
8. Ra Y.G. and Rundensteiner E.A. A transparent schema-evolution system based on object-oriented viewtechnology. *IEEE Trans. Knowl. Data Eng.*, 9(4):600–624, 1997.
9. Roddick J.F. SQL/SE – a query language extension for databases supporting schema evolution. *ACM SIGMOD Rec.*, 21(3):10–16, 1992.
10. Schneiderman B. and Thomas G. An architecture for automatic relational database system conversion. *ACM Trans. Database Syst.*, 7(2):235–257, 1982.
11. Sjøberg D. Quantifying schema evolution. *Inf. Softw. Technol.*, 35(1):35–44, 1993.
12. Tan L. and Katayama T. Meta operations for type management in object-oriented databases – a lazy mechanism for schema evolution. In Proc. First Int. Conf. on Deductive and Object-Oriented Databases, 1989, pp. 241–258.
13. de Vries D. and Roddick J.F. The case for mesodata: an empirical investigation of an evolving database system. *Inf. Softw. Technol.*, 49(9–10):1061–1072, 2007.

Schema Evolution in Process Management Systems

- ▶ Workflow Evolution

Schema Evolution in Workflow Management Systems

- ▶ Workflow Evolution

Schema Mapping

ARIEL FUXMAN¹, RENÉE J. MILLER²

¹Microsoft Research, Mountain View, CA, USA

²University of Toronto, Toronto, ON, Canada

Synonyms

Mapping

Definition

The problem of establishing associations between data structured under different schemas is at the core of many data integration and data sharing tasks. *Schema mappings* establish semantic connections between schemas. Given a source schema S and a target schema T , a *schema mapping* \mathcal{M} is a specification of a relation between instances of S and instances of T . Given an instance of the source I and an instance of the target J that satisfy the mapping, say that $(I, J) \models \mathcal{M}$. Research on schema mapping has focused on the formal specification of schema mappings, the semantics of mappings, along with techniques for creating schema mappings.

Historical Background

Schema mappings have been developed primarily to solve two different problems, each of which has led to a substantial body of research: *data integration* [11] and *data exchange* [4]. In both problems, one is given a source schema S (or a set of source schemas) and an instance I of S , along with a target schema T , which is sometimes called a global schema. A user knows the schema of the target and would like to retrieve source data by posing queries on the target. In data integration, queries posed on the global schema are translated, at query time, to the schema(s) of the local data source(s) and answered using source data. Data integration is sometimes called *virtual data integration* to emphasize that the translated source data are not materialized in the target. In contrast, in data exchange the goal is to translate the source data into a target instance that conforms to the target schema and reflects the source

data as accurately as possible. Target queries are then answered using the materialized target instance.

For both problems, schema mappings are used to describe the semantic relationship between the schemas and their instances, and to determine how queries are translated (in data integration) and what is the best target instance to materialize (in data exchange).

Foundations

Semantics of Schema Mappings

Schema mappings establish semantic connections between schemas. These connections can be represented formally using logical formulas. Assume the existence of two schemas, called the source schema S , and the target schema T . The specifications of S and T may include a set of constraints that instances of the schema must satisfy. A *schema mapping* M is a specification of a relation between instances of S and instances of T . Given an instance of the source I and an instance of the target J that satisfy the mapping, one could say that $(I, J) \models M$. A *mapping setting* is a triple $\mathcal{S} = \langle S, T, M \rangle$. In order to give semantics to a mapping setting, it is assumed that an instance of the source is given, and the goal is to reason about the instances of the target that satisfy the constraints imposed by the schema mapping. That is, given an I , we would like to reason about all J such that $(I, J) \models M$. These target instances are called *solutions*. (This terminology comes from the data exchange literature, and was introduced by Fagin et al. [4]. However, the same concepts are equally applicable to data integration, where the target schema may be referred to as the global or mediated schema.).

Definition

[solution] Let $\mathcal{S} = \langle S, T, M \rangle$ be a mapping setting. Let I be an instance over the source schema S . One could say that an instance J over the target schema T is a *solution* for I in \mathcal{S} if $\langle I, J \rangle \models M$.

A mapping may be a function which defines a single target instance J for each source instance I . If T consists of a single relation, then any view over S defines such a function. Alternatively, in many industrial mapping tools, a mapping is a program which, given an instance of S , outputs an instance of T . But in general, a mapping does not need to be a function and there are clear advantages in having a declarative specification language for mappings. Declarative mapping

specifications permit easier reasoning about the relationship between mappings, something that is essential in designing, maintaining, and evolving mappings.

The most commonly used specification for schema mappings are source-to-target tuple-generating-dependencies (TGDs) which have the form

$$\forall \mathbf{x} (\phi_S(\mathbf{x}) \rightarrow \exists \mathbf{y} \psi_T(\mathbf{x}, \mathbf{y})),$$

where $\phi_S(\mathbf{x})$ is a conjunction of atomic formulas over S and $\psi_T(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over T . TGDs have been generalized in some approaches to permit more general source (ϕ_S) and target (ψ_T) queries (formulas) in the mapping.

Since a solution is any target instance that satisfies the mapping, there may be more than one solution for a given source instance. This fact must be accounted for in the semantics of query answering. The prevalent semantics adopted in the literature is based on the notion of *certain answers*. This semantics takes the conservative approach of returning only the answers that are valid in *every* solution.

Definition

[certain answer] Let \mathcal{S} be a mapping setting. Let I be a source instance such that there exists some solution for I in \mathcal{S} . Let q be a query. We say that a tuple \mathbf{t} is a *certain answer* to q in \mathcal{S} , denoted $\mathbf{t} \in \text{certain}(q, I, \mathcal{S})$, if for every solution J for I in \mathcal{S} , it is the case that $\mathbf{t} \in q(J)$.

In addition to certain answers, other semantics have been explored in the literature, such as epistemic interpretations and probabilistic notions.

To illustrate the notions introduced so far, let S be a schema with relation symbol `Country`(`person, country`). Let T be a schema with relation symbols `Home`(`person, city`) and `Loc`(`city, country`). As an example of a mapping setting, let $\mathcal{S} = \langle S, T, M \rangle$, where M consists of the following source-to-target TGD:

$$\begin{aligned} \forall p, \text{cou}.(S.\text{Country}(p, \text{cou}) \rightarrow \\ \exists \text{cit}.T.\text{Home}(p, \text{cit}) \wedge T.\text{Loc}(\text{cit}, \text{cou})) \end{aligned}$$

There may be more than one solution for a given instance I . For example, let $I = \{\text{Country(john, canada)}\}$. Consider $J_1 = \{\text{Home(john, toronto)}, \text{Loc(toronto, canada)}\}$ and $J_2 = \{\text{Home(john, montreal)}, \text{Loc(montreal, canada)}\}$. It is easy to see that both J_1 and J_2 are solutions for I in \mathcal{S} . The reason for this is that the mapping states that the people and countries

of the source must be in the target, but the city is left unspecified.

Now, consider a query q_1 that retrieves all people from the database. Let $q_1(p) = \exists cit : T.Home(p, cit)$. Since \mathcal{M} must be satisfied by all solutions, there are tuples $Home(john, c)$ and $Loc(c, canada)$ for some city c , in every solution for I . Thus, $(john) \in q_1(J)$, for every solution J . Say that $(john)$ is a certain answer to q_1 , and denote this by $(john) \in \text{certain}(q_1, I, \mathcal{S})$. Next, consider a query q_2 that returns all cities. Let $q_2(cit) = \exists p : T.Home(p, cit)$. In this case, there are no certain answers to q_2 . To see why, notice that there is no tuple t such that $t \in q_2(J_1) \cap q_2(J_2)$. The intuition is that John is the only person in the database, but different solutions may assign him a different city (as long as it is within Canada).

Types of Schema Mappings

Sound, Complete, and Exact Mappings

A common assumption in the data integration and data exchange literature is that the mappings consist of implications, where each side of the implication contains relation symbols coming from the same schema. This results in the following three types of mappings [6,11]. Let ϕ_s be a formula over the source schema, and ψ_t be a formula over the target schema. *Sound mappings* are rules of the form $\phi_s(x) \rightarrow \psi_t(x)$; *complete mappings* are of the form $\psi_t(x) \rightarrow \phi_s(x)$; and *exact mappings* are of the form $\phi_s(x) \leftrightarrow \psi_t(x)$. (Variable quantifiers are omitted for generality.)

There is a substantial body of work on sound mappings. Such systems are sometimes called *open* because the mapping specifies what source data *must* be in the solutions, but it does not give negative information (i.e., it does not specify what *must not* be in any solution). A setting \mathcal{S} containing only sound mappings is referred to as an *open mapping setting*. If J is a solution for an instance I in an open setting \mathcal{S} , and J' is such that $J \subseteq J'$, then J' is a solution for I in \mathcal{S} . As an example, consider an open setting with the following sound mapping.

$$\begin{aligned} \forall p, cou. S.Country(p, cou) \rightarrow \\ \exists cit. T.Home(p, cit) \wedge T.Loc(cit, cou) \end{aligned}$$

Let $I = \{Country(john, canada)\}$. Let $J = \{Loc(calgary, canada), Loc(john, ottawa), Loc(ottawa, canada)\}$, which is a solution for I in \mathcal{S} . The tuple $Loc(calgary,$

$canada)$ does not seem to be related to the sources, and the mapping does not force its addition to the solution. However, it does not forbid its inclusion in the solution either. In fact, one could add any arbitrary tuple to J , and still have a solution for I .

Research on data integration and data exchange has focused primarily on open settings. One reason for this is that, given a source instance, open settings always have a solution. From a practical standpoint, open settings are better suited than settings containing exact or complete mappings in dynamic or autonomous environments, where new sources may be added independently of other sources. With open settings, sources can be described without requiring any knowledge of the other sources, or their relationship to the target. In particular, consider the problem of adding a new data source to an existing data integration (or exchange) system. With open settings, it is not necessary to change any of the existing mappings. A mapping for a new data source cannot conflict with existing mappings.

In contrast, in mapping settings containing complete or exact rules, the addition of a new source may lead to conflicts resulting in a setting for which there may be no solutions. In fact, even for a single setting the use of complete or exact mappings may sometimes preclude the existence of a solution. The problem of deciding the existence of a solution for mapping settings has been studied by Fuxman et al. [5]. As an example of a case in which there may be no solution, consider a setting $\mathcal{S} = \langle S, T, \mathcal{M} \rangle$, where \mathcal{M} consists of the following rules.

$$\begin{aligned} \forall p, cou. S.Country(p, cou) \rightarrow \\ \exists cit. T.Home(p, cit) \wedge T.Loc(cit, cou) \end{aligned} \quad (1)$$

$$\begin{aligned} \forall p, cit, cou. T.Home(p, cit) \wedge T.Loc(cit, cou) \rightarrow \\ S.Capital(cit, cou) \end{aligned} \quad (2)$$

For the source instance $I = \{Country(john, canada), Capital(washington, us)\}$, there is no solution in \mathcal{S} . To see this, assume that there is a solution J for I in \mathcal{S} . By rule (1) of \mathcal{M} , J has tuples $Home(john, c)$ and $Loc(c, canada)$, for some city c . By rule (2) of \mathcal{M} , I is required to have a tuple $Capital(canada, c)$ which it does not. Intuitively, there is no solution for I in \mathcal{S} since the source has no information on what city is the capital of Canada. Notice the effect of rules (1) and (2) on the solutions. The former is a sound mapping and specifies

what *must* be in the solutions; the latter is a complete mapping and constrains what can be in the solutions.

Global-as-View and Local-as-View

In many data integration systems, each relation (or element) of the target schema is defined in terms of the source schemas. In many early systems, there was also an implicit assumption that the schema mapping was a function (for example a view). This approach is known as *global-as-view*. An alternative approach, known as *local-as-view*, was later proposed in which each relation of the source schema is defined in terms of the target schema. For relational systems, these notions can be defined formally as follows [11].

- In *global-as-view* systems (GAV), mappings are of the form $\forall \mathbf{x}. \phi_s(\mathbf{x}) \leftrightarrow R_t(\mathbf{x})$, where R_t is a relation symbol from T and $\phi_s(\mathbf{x})$ is a formula over S (i.e., rather than a formula, there is just one atom on the right-hand-side with no repeated variables). (Note that GAV mappings as originally defined, were typically exact mappings, though more recently sound GAV mappings have also been studied [11].)
- In *local-as-view* systems (LAV), mappings are of the form $\forall \mathbf{x}. R_s(\mathbf{x}) \rightarrow \psi_t(\mathbf{x})$, where R_s is a relation symbol from S and $\psi_t(\mathbf{x})$ is a formula over T (i.e., rather than a formula, there is just one atom on the left-hand-side with no repeated variables).

Recall that queries are posed in terms of the target (global) schema. For this reason, query answering in GAV is easy: the mapping indicates explicitly how to retrieve data from the sources. In particular, given a query q over the target schema, it suffices to *unfold* q using the mapping in order to obtain a rewriting q' of q (i.e., a query that computes the certain answers to q). As an example, consider the following GAV setting $\mathcal{S} = \langle S, T, M \rangle$, where M consists of the following rule.

$$\begin{aligned} \forall cit, cou \exists p. S.Capital(cit, cou) \wedge \\ S.Country(p, cou) \leftrightarrow T.Loc(cit, cou) \end{aligned}$$

Suppose that a user wants to retrieve the cities and countries from relation \texttt{LOC} . Thus, she issues the following query over the global schema:

$$q(cit, cou) = T.Loc(cit, cou)$$

In order to obtain a rewriting of q , it suffices to replace $\texttt{Loc}(cit, cou)$ by its definition in M :

$$\begin{aligned} q'(cit, cou) = \exists p. S.Capital(cit, cou) \wedge \\ S.Country(p, cou) \end{aligned}$$

Query processing in LAV is more involved than in GAV. The reason is that queries are written over the target schema, but a LAV mapping associates views to relations of the source schemas. Thus, the unfolding strategy no longer works in this case; and it is not immediate how to rewrite queries over the source schema. In general, the complexity of query answering in LAV is higher than in GAV. The intuitive explanation is that in GAV, given a source instance I , it suffices to concentrate on a single solution J , whereas in LAV there may be many such solutions.

Rather than unfolding, the problem of reformulating a target query using a LAV mapping boils down to the problem of *answering queries using views* [12].

For many classes of open schema mappings (including source-to-target TGDs) query answering has tractable *data complexity*. For example, query answering is tractable when the schema mappings and queries are conjunctive. In contrast, if complete or exact rules are allowed, the same problem becomes coNP-complete. To show the jump in complexity, consider Figs. 1 and 2 (due to Abiteboul and Duschka [1]). The former gives results for open LAV mappings. The latter gives results under the “closed world assumption”, where mappings consist of exact LAV mappings of the form $\forall \mathbf{x} : R_s(\mathbf{x}) \leftrightarrow \psi_t(\mathbf{x})$, where $R_s(\mathbf{x})$ is a relation from the source, and $\psi_t(\mathbf{x})$ is a formula over the target. The results are given for different logical languages used for the mappings and queries: conjunctive queries (CQ), conjunctive queries with inequalities (CQ^\neq), union of conjunctive queries (UCQ), Datalog, and first-order logic (FO). In addition to these results, the problem of query answering in open mapping settings has also been studied for other query languages (e.g., description logics) and on other data models (e.g., semi-structured models).

mapping	query				
	CQ	CQ^\neq	UCQ	Datalog	FO
CQ	PTIME	coNP	PTIME	PTIME	undec.
CQ^\neq	PTIME	coNP	PTIME	PTIME	undec.
UCQ	coNP	coNP	coNP	coNP	undec.
Datalog	coNP	undec.	coNP	undec.	undec.
FO	undec.	undec.	undec.	undec.	undec.

Schema Mapping. Figure 1. Complexity of query answering using open LAV mappings.

mapping	CQ	$CQ^\#$	query UCQ	Datalog	FO
CQ	coNP	coNP	coNP	coNP	undec.
$CQ^\#$	coNP	coNP	coNP	coNP	undec.
UCQ	coNP	coNP	coNP	coNP	undec.
Datalog	undec.	undec.	undec.	undec.	undec.
FO	undec.	undec.	undec.	undec.	undec.

Schema Mapping. Figure 2. Complexity of query answering using exact LAV mappings.

Schema Mappings in Peer Data Sharing

The success of peer-to-peer (P2P) technology in the domain of file exchange motivated the research community to consider peer-to-peer architectures for data sharing. In a P2P system, participants (peers) rely on one another for service, blurring the distinction between clients and servers, source and target. P2P systems are founded on the principles of *peer autonomy* and *decentralized coordination*. As a result, peers do not have a global view of the system. Rather, global behavior emerges from local interactions.

In a Peer Data Management System (PDMS), each peer has a schema that describes the structure of its data, and can establish connections (typically specified as schema mappings) with other peers in order to exchange data [2,7]. A PDMS is expected to satisfy the desirable properties of P2P systems. For example, the requirement of decentralized coordination precludes the existence of a central catalog. Rather, knowledge about schemas and mappings should be distributed among the peers.

A PDMS should support an arbitrary network of mappings among peers. More importantly, it should be able to exploit the transitive relationships of the network during query answering. A PDMS is essentially a directed graph whose nodes are individual mapping settings, and whose arcs correspond to mappings that relate the schemas in the network. More precisely, there is an arc from P_1 to P_2 if there is a sound rule in some peer setting $\langle P_1, P_2, \mathcal{M} \rangle$ or a complete rule in some peer setting $\langle P_2, P_1, \mathcal{M} \rangle$. It turns out that the topology of this graph has a direct impact on query answering. In particular, Halevy et al. [7] showed the undecidability of the problem of obtaining the certain answers for a PDMS of arbitrary topology, where conjunctions are used for the mapping rules and the queries. Contrast this to the case of a single mapping setting with exact mappings, which is not undecidable, but

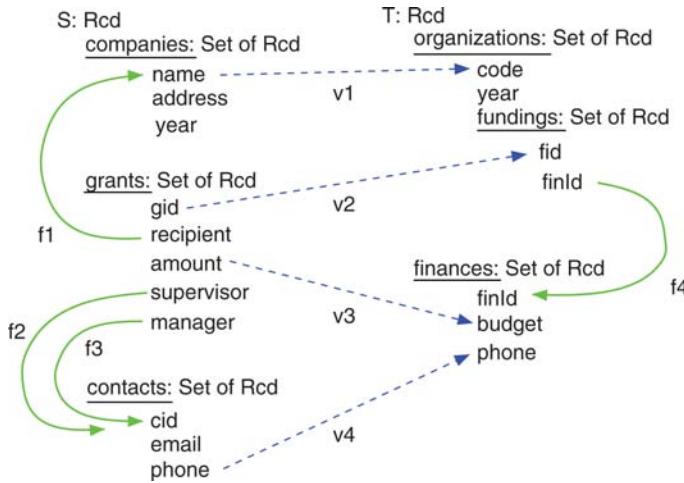
coNP-complete and tractable for open settings. Fuxman et al. [5] studied this problem for a special case of PDMS, called Peer Data Exchange, and gave a class of mappings and queries for which the problem is decidable under the existence of cycles. Calvanese et al. [3] proposed an alternative semantics for query answering, based on epistemic interpretations, for which obtaining the certain answers in a PDMS of arbitrary topology is decidable and, in some cases, tractable. An alternative approach involves specifying schema mappings using *mapping tables* which specify how data values are mapped between peers [10]. Research is on-going on what are good mapping formalisms to support PDMS.

Creating Schema Mappings

Creating a schema mapping between independently designed schemas can be a tremendous challenge. Schemas that are designed independently, even if they represent the same or similar information may use different names and structures to describe the same or similar data. Designing schema mappings by hand is known to be a very difficult task requiring expert users familiar with both source and target schemas. Even experts can often make errors leading to specifications that omit information or produce incorrect answers to target queries.

To help automate this task, Milo and Zohar proposed the use of *schema matchings* (or matchings) which indicate potential associations between elements within different schemas [14]. A matching is most often represented as a set of pairs of schema attributes from two different schemas. Schema matchings can be semi-automatically inferred by using a variety of matching tools. These tools use schema and data characteristics such as lexical similarities, structural proximity, data values, etc. to infer potential matches between attributes of different schemas. Matchings, however, do not represent the full semantic relationship between schemas and their instances.

Figure 3 shows two schemas that represent information about companies and grants. The left-most schema (the source S) is a relational schema (with three tables *companies*, *grants*, and *contacts*), presented in a nested relational representation that is used as a common platform for modeling relational and XML schemas. The curved lines $f1$, $f2$ and $f3$ in the figure represent either foreign keys or simple inclusion dependencies, specified as part of the schema (or discovered using a dependency or constraint miner).



Schema Mapping. Figure 3. A matching between source and target schemas.

The schema on the right (the target T), records the funding (*fundings*) that an organization (*organizations*) receives, nested within the respective organization element. The amount of each funding is recorded in the *finances* record along with a contact phone number (*phone*).

Figure 3 indicates, using the dotted lines v_1 through v_4 , a matching entered by a schema expert or discovered by a matching tool. Due to the heterogeneity and the different requirements under which the two databases were developed, the same real world entity (for example, a company ‘IBM’) may be represented in very different ways in the two databases, and structures that appear to be the same may actually model different concepts. In our example, the matching v_1 indicates that what is called a company *name* in the first schema, is referred to as an organization *code* in the second. On the other hand, both schemas have an element *year*, but there is no match between these attributes indicating that they likely do not represent the same concept. For instance, element *year* in the source schema may represent the time the company was founded, while in the target it may represent the time the company had its initial public offer.

Note that matchings are far from sufficient to tell us how the data instances of S and T are related. While the matching may indicate that *companies.name* data should appear in the *organizations.code* attribute of the target and that *grants.gid* data should appear in *fundings.fid*, it does not tell us which grant should be associated with which company. Similarly, if one relies solely on the matching, one could map *grants*.

to *fundings.fid* and leave the *finId* attribute value empty (since there is no matching for *finId*). If one does this, in the target data, there will be no way to associate a funding record with a finance record. However, the foreign key f_4 indicates there is a real world relationship between these concepts.

The generation of schema mappings have been considered in a number of research projects and industrial tools. The Clio project [13] was the first mapping system to exploit logical reasoning about the semantics embedded in the schemas and their instances to help automate mapping creation. In this work, the mapping discovery process has been referred to as *query discovery* in that the goal is to discovery a query over the source (ϕ_S), a query over the target (ψ_T), and their relationship, in order to create a set of possible source-to-target TGDs: $\forall x(\phi_S(x) \rightarrow \exists y\psi_T(x, y))$ [15]. These mappings can then be shown to a mapping designer (visually or using data examples) who can decide if they correctly represent the relationship between source and target instances.

To illustrate Clio’s approach, consider our example schemas. In the target, the nesting structure within *organizations* indicates that there is a real-world relationship between organizations and their fundings— that is, the association of a specific funding record with a specific organization has some natural semantics in the domain. For example, the nesting may represent fundings given to (or alternatively given by) an organization. Hence, in creating a mapping, Clio will consider related associations in the source. In the example, the data that matches *organizations* and

fundings comes from companies and grants (via the matches v_1 and v_2). There is a relationship between these two tables represented by the inclusion dependency on grants.recipient. Hence, by ignoring v_3 and v_4 for the moment, Clio will suggest the following source-to-target TGD to a user. (Notation is slightly abused and let \underline{F} represent identifiers for sets nested inside of organizations.)

$$\begin{aligned} \forall n, \forall d, \forall y, \forall g, \forall a, \forall s, \forall m \\ \text{companies}(n, d, y), \text{grants}(g, n, a, s, m) \rightarrow \\ \exists y', \underline{F}, f, \text{organizations}(n, y', \underline{F}), \underline{F}(g, f) \end{aligned} \quad (3)$$

Notice that this mapping retains the semantic association between companies and their grants, and uses this to associate organizations with a set of related fundings. This mapping is correct if these two associations represent the same real world association, something a user must verify. By extending this example to include v_3 , Clio will see that there is an association between gid and amount in the source (because these values are paired in the same record) and consider possible ways of associating the matched attributes (in this case fid and budget) in the target. These attributes are not in the same record in the target, but are associated through a foreign key on finId. To maintain the source association in the target, Clio creates a mapping containing a target join on finId. Finally, if one considers how to create finances tuples in the target, notice that there are two possible ways of associating the related source data (grants.amount and contacts.phone) – using a join on supervisor and cid, or using a join on manager and cid. These joins represent different semantic associations and Clio will create mappings corresponding to each and let the user decide which (if any) of these associations should be preserved in the target data. One of the mappings created by Clio, which uses the source association represented by f_2 , is illustrated below.

$$\begin{aligned} \forall n, \forall d, \forall y, \forall g, \forall a, \forall s, \forall m, \forall e, \forall p \text{ companies}(n, d, y) \\ \wedge \text{grants}(g, n, a, s, m) \wedge \text{contacts}(s, e, p) \rightarrow \\ \exists y', \underline{F}, f, \text{organizations}(n, y', \underline{F}) \wedge \underline{F}(g, f) \wedge \\ \text{finances}(f, a, p) \end{aligned} \quad (4)$$

Clio's mapping discovery algorithm is based on an extension of standard relational dependency inference (based on the chase) to nested relational schemas. The schemas may be relational or nested relational

containing source and target TGDs (e.g., inclusion dependencies) and egds (e.g., functional dependencies).

Data Translation

Mapping tools that create declarative mappings provide a way of translating these specifications into programs (transformation code) that given a source instance produce a single target instance for data exchange [15,16]. Industrial mapping systems, such as Altova Mapforce (http://www.altova.com/products/mapforce/data_mapping.html), Stylus Studio (<http://www.stylusstudio.com>), or Aqualogic (<http://www.bea.com/aqualogic>) are often visual programming systems which compile visual specifications of mappings into executable code including SQL, XSLT, Java or C.

Within this area, there has been a great deal of work on producing data transformation code that is modular and efficient [9,16]. For schema mappings that permit many solutions, a decision must be made as to what is the “best” solution to materialize. Notice that there may be target data (for example, organizations.year, or fundings.finId attributes from Fig. 3) that do not correspond to any source data. It may not be sufficient to simply fill in null values for this information. Consider the fundings.finId and finances.finId from Fig. 3. If one fills both with null values, it is possible to join fundings with finances to find the budget of a specific funding. As an alternative, to maintain the association between the source values grants.gid and grants.amount as this data are translated into the target, one option is to create identifiers (using Skolem functions) that represent the desired association [8]. The Clio project was the first to consider systematically how to create Skolem functions that fill in missing target data specifically for data exchange [15].

Key Applications

Schema mappings are foundational to enabling data integration, data exchange, schema evolution, and data translation (between data models). Applications of schema mappings include Enterprise Information Integration (EII), e-commerce, object-to-relational wrappers, XML-to-relational mapping, data warehousing, and portal design tools.

Cross-references

- [Answering Queries Using Views](#)
- [Certain Answers](#)
- [Data Exchange](#)

- ▶ Peer Data Management System
- ▶ Peer-to-Peer Data Integration
- ▶ Schema Mapping Composition
- ▶ Schema Matching

Recommended Reading

1. Abiteboul S. and Duschka O.M. Complexity of answering queries using materialized views. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 254–263.
2. Bernstein P.A., Giunchiglia F., Kementsietsidis A., Mylopoulos J., Serafini L., and Zaihrayeu I. Data management for peer-to-peer computing: a vision. In Proc. 5th Int. Workshop on the World Wide Web and Databases, 2002.
3. Calvanese D., De Giacomo G., Lenzerini M., and Rosati R. Logical foundations of peer-to-peer data integration. In Proc. 23rd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2004, pp. 241–251.
4. Fagin R., Kolaitis P.G., Miller R.J., and Popa L. Data exchange: semantics and query answering. *Theor. Comput. Sci.*, 336(1):89–124, May 2005.
5. Fuxman A., Kolaitis P.G., Miller R.J., and Tan W.-C. Peer data exchange. *ACM Trans. Database Syst.*, 31(4):1454–1498, 2006.
6. Grahne G. and Mendelzon A.O. Tableau techniques for querying information sources through global schemas. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 332–347.
7. Halevy A., Ives Z., Suciu D., and Tatarinov I. Schema mediation in peer data management systems. In Proc. 9th Int. Conf. on Data Engineering, 2003, pp. 505–518.
8. Hull R. and Yoshikawa M. ILOG: declarative creation and manipulation of object identifiers. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 455–468.
9. Jiang H., Ho H., Popa L., and Han W.S. Mapping-driven xml transformation. In Proc. 16th Int. World Wide Web Conf., 2007, pp. 1063–1072.
10. Kementsietsidis A., Arenas M., and Miller R.J. Mapping data in peer-to-peer systems: Semantics and Algorithmic Issues. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 325–336.
11. Lenzerini M. Data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
12. Levy A.Y., Mendelzon A.O., Sagiv Y., and Srivastava D. Answering queries using views. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1995, pp. 95–104.
13. Miller R.J., Haas L.M., and Hernández M. Schema mapping as query discovery. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 77–88.
14. Milo T. and Zohar S. Using schema matching to simplify heterogeneous data translation. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 122–133.
15. Popa L., Velegrakis Y., Miller R.J., Hernández M.A., and Fagin R. Translating web data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 598–609.
16. Shu N.C., Housel B.C., Taylor R.W., Ghosh S.P., and Lum V.Y. EXPRESS: a data eXtraction, processing, and restructuring system *ACM Trans. Database Syst.*, 2(2):134–174, 1997.

Schema Mapping Composition

WANG-CHIEW TAN

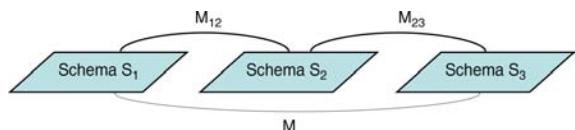
University of California-Santa Cruz, Santa Cruz,
CA, USA

Synonyms

Mapping composition; Semantic mapping composition

Definition

A *schema mapping* (or *mapping*) is a triple $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma)$, where \mathbf{S}_1 and \mathbf{S}_2 are relational schemas with no relation symbols in common and Σ is a set of formulas of some logical formalism over $(\mathbf{S}_1, \mathbf{S}_2)$. An *instance* of \mathcal{M} is a pair (I, J) where I is an instance of \mathbf{S}_1 and J is an instance of \mathbf{S}_2 such that (I, J) satisfies every formula in the set Σ . The set of all instances of \mathcal{M} is denoted as $\text{Inst}(\mathcal{M})$.



Let $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ be two consecutive mappings such that there are no relation symbols in common between any two schemas of \mathbf{S}_1 , \mathbf{S}_2 and \mathbf{S}_3 . A mapping $\mathcal{M} = (\mathbf{S}_1, \mathbf{S}_3, \Sigma)$ is a *composition* of \mathcal{M}_{12} and \mathcal{M}_{23} if $\text{Inst}(\mathcal{M}) = \text{Inst}(\mathcal{M}_{12}) \circ \text{Inst}(\mathcal{M}_{23})$. In other words, $\text{Inst}(\mathcal{M})$ is the set of all pairs (I, J) such that I is an instance of \mathbf{S}_1 , J is an instance of \mathbf{S}_3 and there exists an instance K of \mathbf{S}_2 such that $(I, K) \in \text{Inst}(\mathcal{M}_{12})$ and $(K, J) \in \text{Inst}(\mathcal{M}_{23})$.

Historical Background

Mappings are widely used in the specification of relationships between data sources in applications such as data integration, data exchange, and peer data management systems. The model management framework introduced by Bernstein et al. [3], where the primary abstractions are models and mappings between

models, can be used to model these applications. Several operators for manipulating mappings between models were introduced in this framework. Among them, the *composition operator* is one of the most fundamental operators for manipulating mappings between models.

Madhavan and Halevy [10] first studied the problem of composing mappings between relational schemas (mappings are also called *semantic mappings* in [10]). They gave a definition of the semantics of the composition operator in the context where mappings are specified by sound *Global-Local-As-View (GLAV)* formulas [9]. Sound GLAV formulas are the most widely used and studied form of mappings in data integration systems and they are equivalent to *source-to-target tuple generating dependencies (s-t tgds)* [13]. (See Foundations section for a definition of s-t tgds.) The Madhavan and Halevy semantics of composition is different from the one stated above; The set of formulas that specifies the composition \mathcal{M} of two successive mappings, \mathcal{M}_{12} and \mathcal{M}_{23} , is relative to a class of queries Q that is defined over the schema S_3 . This means that for every query q in Q , the certain answers of q according to \mathcal{M} coincide with the certain answers of q that would be obtained by applying the consecutive mappings \mathcal{M}_{12} and \mathcal{M}_{23} .

The Madhavan and Halevy notion of composition is termed a composition that is *certain answer adequate* for Q in [8]. Fagin et al. [8] showed that while certain answer adequacy may be sufficient for obtaining the certain answers of any query in Q , whether via Madhavan and Halevy's notion of composition \mathcal{M} or through the successive mappings \mathcal{M}_{12} and \mathcal{M}_{23} , there may be formulas that are logically inequivalent to Σ of \mathcal{M} that are also certain answer adequate for Q . In other words, there can be two mappings \mathcal{M} and \mathcal{M}' that are both certain answer adequate for the consecutive mappings \mathcal{M}_{12} and \mathcal{M}_{23} with respect to the class of queries Q , but the sets of formulas Σ and Σ' of \mathcal{M} and \mathcal{M}' , respectively, are logically inequivalent. It was also shown in [8] that certain answer adequacy is a rather fragile notion. A mapping \mathcal{M} that is a composition of \mathcal{M}_{12} and \mathcal{M}_{23} with respect to a class Q of conjunctive queries may no longer be a composition of the same two mappings when Q is extended to the class of conjunctive queries with inequalities.

Fagin et al. [8] introduced a definition of composition that is based entirely on the set-theoretic composition of instances of the two successive mappings \mathcal{M}_{12}

and \mathcal{M}_{23} . (See the Definition section for the set-theoretic definition given in [8].) Unlike the definition of composition given by Madhavan and Halevy, Fagin et al.'s [8] definition is not relative to a class of queries. Furthermore, the set of formulas in \mathcal{M} that defines the composition of \mathcal{M}_{12} and \mathcal{M}_{23} is unique up to logical equivalence. Hence, with Fagin et al.'s [8] notion of composition, \mathcal{M} is referred to as *the* composition of \mathcal{M}_{12} and \mathcal{M}_{23} . It was also shown in [8] that the composition \mathcal{M} is always certain answer adequate for \mathcal{M}_{12} and \mathcal{M}_{23} for *every* class of queries.

The results established in [8] were based on mappings specified by s-t tgds. In other words, Fagin et al. [8] assumes that the sets of formulas Σ_{12} and Σ_{23} in the successive mappings \mathcal{M}_{12} and \mathcal{M}_{23} , respectively, are finite sets of s-t tgds. A subsequent paper by Nash et al. [12] also studied the composition operator where mappings are specified by *embedded dependencies*. Embedded dependencies are more general than s-t tgds and can model constraints such as keys. Among the results established in [12] is an algorithm that computes the composition of two successive mappings specified by embedded dependencies. Their algorithm may not terminate in general and the authors characterized sufficient conditions on the input mappings for which their composition algorithm is guaranteed to produce a composition of the input mappings. An implementation of the composition operator that extends the composition algorithm of [12] is described in [2].

Foundations

In [8], mappings are specified by finite sets of s-t tgds which are equivalent to sound GLAV assertions used in [10]. A *s-t tgd* is a first-order formula of the form:

$$\forall \mathbf{x}(\phi_S(\mathbf{x}) \rightarrow \exists \mathbf{y}\psi_T(\mathbf{x}, \mathbf{y})),$$

where $\phi_S(\mathbf{x})$ is a conjunction of atomic formulas over the schema S and where $\psi_T(\mathbf{x}, \mathbf{y})$ is a conjunction of atomic formulas over the schema T . Every variable in \mathbf{x} and \mathbf{y} must appear in ϕ_S and ψ_T respectively. However, some variables in \mathbf{x} need not appear in ψ_T . A *full source-to-target tuple generating dependency (full s-t tgd)* is a special s-t tgd of the form

$$\forall \mathbf{x}(\phi_S(\mathbf{x}) \rightarrow \psi_T(\mathbf{x})),$$

where no existentially-quantified variables appears on the right-hand-side of the s-t tgd. As before, $\phi_S(\mathbf{x})$ is a

conjunction of atomic formulas over S and $\psi_T(y)$ is a conjunction of atomic formulas over T . Every variable in x must appear in ϕ_S .

Example 1. Let $\mathcal{M}_{12} = (S_1, S_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (S_2, S_3, \Sigma_{23})$ be two successive mappings. The schema S_1 consists of a binary relation symbol `Takes`, the second schema S_2 consists of two binary relation symbols `Takes` and `Student`, and the third schema S_3 consists of a single binary relation symbol `Enrollment`. The formulas in Σ_{12} and Σ_{23} are s-t tgds stated below:

$$\begin{aligned}\Sigma_{12} = & \{\forall_n \forall_c (\textbf{Takes}(n, c) \rightarrow \textbf{Takes}_1(n, c)), \\ & \forall_n \forall_c (\textbf{Takes}(n, c) \rightarrow \exists_s \textbf{Student}(n, s))\}\end{aligned}$$

$$\begin{aligned}\Sigma_{23} = & \{\forall_n \forall_s \forall_c (\textbf{Student}(n, s) \wedge \textbf{Takes}_1(n, c)), \\ & \textbf{Enrollment}(s, c))\}\end{aligned}$$

Observe that the first s-t tgd in Σ_{12} and the s-t tgd in Σ_{23} are full s-t tgds. The s-t tgds in Σ_{12} state that the `Takes` relation in S_2 contains the `Takes` relation in S_1 and that every student with name n who takes a course c (in `Takes`) has a associated tuple in `Student` with name n and some student id s . The s-t tgd in Σ_{23} states that every student with name n and student id s (in `Student`) who is also taking a course c (in `Takes`) must have a corresponding tuple (s, c) that associates the student id s with the course c in the `Enrollment` relation of S_3 .

Recall from the definition that the composition \mathcal{M}_{12} and \mathcal{M}_{23} is a mapping \mathcal{M} that captures exactly the set of instances $\text{Inst}(\mathcal{M}_{12}) \circ \text{Inst}(\mathcal{M}_{23})$. The set of instances $\{(I_1, I_3) \mid (I_1, I_3) \in \text{Inst}(\mathcal{M}_{12}) \circ \text{Inst}(\mathcal{M}_{23})\}$ is called the *composition query* of \mathcal{M}_{12} and \mathcal{M}_{23} . Among the issues investigated by Fagin et al. [8] in composing mappings under this semantics are:

- Is the language of s-t tgds always sufficient to define the composition of two successive mappings?
- What is the complexity of the instances associated with the composition query of two successive mappings?
- What is a right language for composing mappings?
- How does data exchange and query answering behave in the chosen language for composing mappings?

Composing s-t TGDs: Definability and Complexity.

The answer to the first question above is no. It was shown in [8] that if \mathcal{M}_{12} and \mathcal{M}_{23} are specified by

finite sets of full s-t tgds and s-t tgds respectively, then the composition of \mathcal{M}_{12} with \mathcal{M}_{23} is always definable by a finite set of s-t tgds. Furthermore, the associated composition query is in PTIME. However, if \mathcal{M}_{12} is specified by a set of s-t tgds, not necessarily full, then the composition of \mathcal{M}_{12} with \mathcal{M}_{23} may not always be definable by a finite set of s-t tgds. For instance, the composition of the successive mappings in Example 1 is not definable by any finite set of s-t tgds. However, the composition of \mathcal{M}_{12} and \mathcal{M}_{23} is definable by an infinite set of s-t tgds and, in fact, definable by a first-order formula. As a consequence, the composition query of \mathcal{M}_{12} and \mathcal{M}_{23} is a PTIME query.

Fagin et al. [8] further showed that there exists successive mappings \mathcal{M}_{12} and \mathcal{M}_{23} where the composition query of \mathcal{M}_{12} and \mathcal{M}_{23} is NP-complete and the composition of the two mappings is not definable in least fixed-point logic LFP. The mappings used are such that \mathcal{M}_{12} is specified by a finite set of s-t tgds each having at most one existentially-quantified variable and \mathcal{M}_{23} consists of only one full s-t tgd. Essentially, the NP-hardness result is obtained by a reduction from 3-COLORABILITY to the composition query of two fixed mappings with the above properties. They showed that the composition query of mappings specified by finite sets of s-t tgds is always in NP.

Second-Order TGDs. Since the composition query of mappings specified by finite sets of s-t tgds is always in NP, it follows from Fagin's theorem [5] that the composition of the two mappings is always definable by an existential second-order formula, where the existential second-order variables are interpreted over relations on the union of the set of values in I_1 with the set of values in I_3 . Here, $(I_1, I_3) \in \text{Inst}(\mathcal{M}_{12}) \circ \text{Inst}(\mathcal{M}_{23})$. Fagin et al. [8] showed that, in fact, the composition of mappings specified by finite sets of s-t tgds is always definable by a restricted form of existential second-order formula, called *second-order tgds* (SO tgds).

SO tgds are s-t tgds that are extended with existentially quantified functions and with equalities. SO tgds are the “right” language for composing mappings because they form the smallest well-behaved extension to the class of s-t tgds that is closed under conjunction and composition. In addition, as explained later, SO tgds also possess good properties for data exchange and query answering. The precise definition of SO tgds is given next, after the definition of *terms*.

Given a collection x of variables and a collection f of function symbols, a *term* (based on x and f) is

defined recursively as follows: (i) Every variable in \mathbf{x} is a term. (ii) If f is a k -ary function symbol in \mathbf{f} and t_1, \dots, t_k are terms, then $f(t_1, \dots, t_k)$ is a term. Let \mathbf{S} be a source schema and \mathbf{T} a target schema. A *second-order tgd* (*SO tgd*) is a formula of the form:

$$\exists \mathbf{f}((\forall \mathbf{x}_1(\phi_1 \rightarrow \psi_1)) \wedge \dots \wedge (\forall \mathbf{x}_n(\phi_n \rightarrow \psi_n))),$$

where

1. Each member of \mathbf{f} is a function symbol.
2. Each ϕ_i is a conjunction of
 - atomic formulas of the form $S(y_1, \dots, y_k)$, where S is a k -ary relation symbol of schema \mathbf{S} and y_1, \dots, y_k are variables in \mathbf{x}_i , not necessarily distinct, and
 - equalities of the form $t = t'$ where t and t' are terms based on \mathbf{x}_i and \mathbf{f} .
3. Each ψ_i is a conjunction of atomic formulas $T(t_1, \dots, t_l)$, where T is an l -ary relation symbol of schema \mathbf{T} and t_1, \dots, t_l are terms based on \mathbf{x}_i and \mathbf{f} .
4. Each variable in \mathbf{x}_i appears in some atomic formula of ϕ_i .

Each subformula $\forall \mathbf{x}_i(\phi_i \rightarrow \psi_i)$ is a *conjunction* of the SO tgd. The last condition is a safety condition similar to that made for s-t tgds. As an example, the formula $\exists f \forall x \forall y(S(x) \wedge (y = f(x)) \rightarrow T(x, y))$ is not a SO tgd because the safety condition is violated. In particular, the variable y does not appear in an atomic formula on the left-hand-side of the formula. As another example, the formula

$$\exists f \forall n \forall c (\mathbf{Takes}(n, c) \rightarrow \mathbf{Enrollment}(f(n), c))$$

is a SO tgd. Recall that the composition of the successive mappings in Example 1 is not definable by any finite set of s-t tgds. Fagin et al. [8] showed that the SO tgd above defines the composition of the two mappings given in Example 1. Hence, existentially quantified function symbols are a necessary extension to the language of s-t tgds for composing mappings. Intuitively, the SO tgd states that if a student with name n takes a course c in **Takes**, then the student id of n , which is denoted by the function $f(n)$, is associated with c in **Enrollment**. An example of SO tgds where equalities are involved is described next.

Example 2. Let $\mathcal{M}_{12} = (\mathbf{S}_1, \mathbf{S}_2, \Sigma_{12})$ and $\mathcal{M}_{23} = (\mathbf{S}_2, \mathbf{S}_3, \Sigma_{23})$ be two successive mappings. The first schema \mathbf{S}_1 consists of a unary relation symbol **Emp**, the second schema \mathbf{S}_2 consists of a binary relation

symbol **Mgr**, and the third schema \mathbf{S}_3 consists of a binary relation symbol **Mgr** and a unary relation symbol **SelfMgr**. The s-t tgds in Σ_{12} and Σ_{23} are:

$$\sum_{12} = \{\forall e (\mathbf{Emp}(e) \rightarrow \exists m \mathbf{Mgr}_1(e, m))\}$$

$$\begin{aligned} \sum_{23} = & \{\forall e \forall m (\mathbf{Mgr}_1(e, m) \rightarrow \mathbf{Mgr}(e, m)), \\ & \forall e (\mathbf{Mgr}_1(e, e) \rightarrow \mathbf{SelfMgr}(e))\} \end{aligned}$$

Intuitively, the s-t tgd in Σ_{12} asserts that every employee e in **Emp** must have a manager m and this association can be found in **Mgr**. The first s-t tgd in Σ_{23} asserts that the **Mgr** relation contains the **Mgr** relation and the second s-t tgd in Σ_{23} asserts that **SelfMgr** contains employees who are their own managers according to the **Mgr** relation.

Fagin et al. [8] showed that the following SO tgd defines the composition of \mathcal{M}_{12} and \mathcal{M}_{23} .

$$\begin{aligned} \exists f (\forall e (\mathbf{Emp}(e) \rightarrow \mathbf{Mgr}(e, f(e))) \wedge \forall e (\mathbf{Emp}(e) \\ \wedge (e = f(e)) \rightarrow \mathbf{SelfMgr}(e))) \end{aligned}$$

They also showed that the above SO tgd is not logically equivalent to any finite or infinite sets of SO tgds without equalities. In other words, the composition of \mathcal{M}_{12} and \mathcal{M}_{23} is not definable by SO tgds without equality. Hence, equalities are a necessary extension to the language of s-t tgds for composing mappings.

The extensions of s-t tgds with function symbols and equalities in SO tgds are necessary to compose mappings specified by s-t tgds. Fagin et al. [8] also showed that SO tgds are closed under composition. This means that the composition of two mappings, each specified by an SO tgd, is another SO tgd.

Example 3. An illustration of the algorithm described in [8] for composing two mappings, based on Example 2, is described next. The algorithm takes as input two mappings, \mathcal{M}_{12} and \mathcal{M}_{23} , specified by s-t tgds and returns as output a mapping \mathcal{M}_{13} that defines the composition of the two input mappings.

The first step of the algorithm is to transform the s-t tgds in Σ_{12} and Σ_{23} into SO tgds by introducing Skolem functions to replace existentially quantified variables. For example, Σ_{12} and Σ_{23} will now become Σ'_{12} and Σ'_{23} respectively.

$$\sum'_{12} = \{\exists f \forall e (\mathbf{Emp}(e) \rightarrow \mathbf{Mgr}_1(e, f(e)))\}$$

$$\Sigma'_{23} = \{\forall e \forall m (\mathbf{Mgr}_1(e, m) \rightarrow \mathbf{Mgr}(e, m)), \\ \forall e (\mathbf{Mgr}_1(e, e) \rightarrow \mathbf{SelfMgr}(e))\}$$

In particular, observe that Σ'_{12} now consists of an SO tgd with function $f(e)$ that denotes the manager of employee e . The arguments of f consist of all universally quantified variables in the s-t tgd. The next step of the algorithm combines Σ'_{12} with Σ'_{23} to obtain Σ'_{13} by replacing atomic formulas on the left-hand-side of conjuncts of SO tgds in Σ'_{23} with atomic formulas from S_1 through conjuncts of SO tgds in Σ'_{12} . In the running example, $\mathbf{Emp}(e_0) \rightarrow \mathbf{Mgr}_1(e_0, f(e_0))$ is combined with $\mathbf{Mgr}_1(e, m) \rightarrow \mathbf{Mgr}(e, m)$ to obtain

$$\mathbf{Emp}(e_0) \wedge (e = e_0) \wedge (m = f(e_0)) \rightarrow \mathbf{Mgr}(e, m)$$

and $\mathbf{Emp}(e_1) \rightarrow \mathbf{Mgr}_1(e_1, f(e_1))$ is combined with $\mathbf{Mgr}_1(e, e) \rightarrow \mathbf{SelfMgr}(e)$ to obtain

$$\mathbf{Emp}(e_1) \wedge (e = e_1) \wedge (e = f(e_1)) \rightarrow \mathbf{SelfMgr}(e)$$

Observe that the equalities generated by this step has the form $y = t$ where y is a variable in Σ'_{23} and t is a term based on the variables and functions of Σ'_{12} . The next step of the algorithm removes variables from Σ'_{23} according to the equalities. For example, e is replaced with e_0 in the first formula and e is replaced with e_1 in the second formula to obtain the following SO tgds:

$$\mathbf{Emp}(e_0) \wedge (m = f(e_0)) \rightarrow \mathbf{Mgr}(e_0, m)$$

$$\mathbf{Emp}(e_1) \wedge (e_1 = f(e_1)) \rightarrow \mathbf{SelfMgr}(e_1)$$

At this point, the variable m is replaced with $f(e_0)$ to obtain the following SO tgds:

$$\mathbf{Emp}(e_0) \rightarrow \mathbf{Mgr}(e_0, f(e_0))$$

$$\mathbf{Emp}(e_1) \wedge (e_1 = f(e_1)) \rightarrow \mathbf{SelfMgr}(e_1)$$

Finally, when no more variables of Σ'_{23} can be replaced, the following SO tgd is returned.

$$\exists f (\forall e (\mathbf{Emp}(e) \rightarrow \mathbf{Mgr}(e, f(e))) \wedge \forall e (\mathbf{Emp}(e) \\ \wedge (e = f(e)) \rightarrow \mathbf{SelfMgr}(e)))$$

Data Exchange and Query Answering. Let $\mathcal{M} = (S_1, S_2, \Sigma)$ be a mapping. Given a finite instance I over the schema S_1 , the *data exchange problem* [7] is to construct a finite instance J over the schema S_2 such that (I, J) satisfies all the formulas specified in Σ . Such an instance J is called a *solution* of I under

the mapping \mathcal{M} . If Σ is specified by a finite set of s-t tgds, many solutions for I under \mathcal{M} may exist in general because s-t tgds underspecify the data exchange process in general. In [7], the classical *chase* procedure [1,11] has been used to construct *universal* solutions of I under a mapping \mathcal{M} . Universal solutions are the most general type of solutions in the following sense: If J is a universal solution for I under \mathcal{M} , this means that J is a solution for I under \mathcal{M} with the additional property that J has a *homomorphism* into every solution for I under \mathcal{M} . Intuitively, an instance K has a homomorphism into an instance K' if K can be embedded in K' (modulo the renaming of nulls that occur in K). It was shown in [7] that universal solutions can be computed in polynomial time when the mapping is fixed. Universal solutions are desirable not only because they are the most general, but also because they can be used to compute the certain answers of unions of conjunctive queries that are posed against the schema S_2 in polynomial time. If q is a k -ary query over S_2 , then the certain answers of q with respect to an instance I over S_1 , denoted as $\text{certain}_{\mathcal{M}}(q, I)$, is the set of all k -tuples t of constants from I such that $t \in q(J)$ for every solution J of I under \mathcal{M} . It was shown in [7] that if J is a universal solution for I under \mathcal{M} and q is a union of conjunctive queries, then $\text{certain}_{\mathcal{M}}(q, I)$ can be computed as follows: (i) Evaluate $q(J)$ and then (ii) discard tuples from $q(J)$ that contain nulls. The remaining tuples obtained from this process, denoted as $q(J)_\downarrow$, form the certain answers of q with respect to I .

In the case where there are two or more successive mappings specified by s-t tgds and only the target instance over the last schema is of interest, one approach to obtain a universal solution of the last schema when given an instance I over the first schema is to perform a series of data exchanges (using the chase procedure [7]) starting from I according to the sequence of mappings. The final target instance that is arrived through this process is a universal solution for I for the sequence of mappings. (The series of data exchanges produces a universal solution. This result can be found in Proposition 7.2 of [6].) Obviously, one drawback of this approach is the unnecessary construction of potentially many intermediate instances which are not of interest. Another approach that avoids the construction of intermediate instances altogether is to first compose the sequence of mappings to obtain a composed mapping over the first source

schema and the final target schema. After this, data are exchanged (by using the chase procedure) according to the composed mapping. However, as described earlier, the language of s-t tgds may no longer be sufficient for defining the composition of two or more successive mappings. Instead, SO tgds are needed to describe the composition of successive mappings in general. In [8], the classical chase technique is extended to SO tgds. They showed that chasing with SO tgds is again a polynomial time procedure and that the chase with SO tgds produces a universal solution as in s-t tgds. Hence, a universal solution for the final target schema can be computed by simply chasing I over the composed mapping. As a consequence, the certain answers of unions of conjunctive queries that are posed over the target schema of a mapping specified by SO tgds can also be computed in polynomial time: First, a universal solution J of I is computed by chasing I with the mapping. After this, compute $q(J)_{\downarrow}$, as described earlier.

Key Applications

One important application of composing mappings is schema evolution. Consider the figure shown in the Definition section. If S_3 is an evolved schema of S_2 and the mappings M_{12} and M_{23} are given, then it is possible to derive the direct relationships between S_1 and S_3 by composing M_{12} with M_{23} . See [14] for an application of composition to schema evolution.

Another important application of composition is to optimize the migration of data through a sequence of mappings. An end-to-end mapping is first assembled from a sequence of two or more mappings by composition before data are migrated through the assembled mapping. The benefit of using an end-to-end mapping for migrating data from the first schema to the last schema in the sequence of mappings is the potential savings from the sequence of unnecessary data migration steps through the intermediate schemas along the sequence of mappings. For a similar reason, the end-to-end mapping could also be used to optimize query rewriting. Referring back to the figure in the Definition section, if a query that is posed against the schema S_3 needs to be rewritten into a query against the schema S_1 , it is potentially rewarding to first compose the sequence of mappings M_{12} and M_{23} and then reason about the rewriting through the composition rather than through the sequence of mappings M_{23} and M_{12} .

Composed mappings can also be used as an abstraction for a sequence of data migration steps. A recent work [4] on Extract-Transform-Load (ETL) systems illustrates this point. An ETL script can be modeled as a network of mappings describing the flow of data from a source to a target. By composing various sequences of mappings in the network of mappings, an abstraction of the overall ETL transformation can be achieved.

Cross-references

- ▶ [Data Exchange](#)
- ▶ [Schema Mapping](#)

Recommended Reading

1. Beeri C. and Vardi M.Y. A Proof Procedure for Data Dependencies. *J. ACM*, 31(4):718–741, 1984.
2. Bernstein P.A., Green T.J., Melnik S., and Nash A. Implementing Mapping Composition. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 55–66.
3. Bernstein P.A., Halevy A.Y., and Pottinger R. A Vision of Management of Complex Models. *ACM SIGMOD Rec.*, 29(4):55–63, 2000.
4. Dessloch S., Hernández M., Wisnesky R., Radwan A., and Zhou J. Orchid: Integrating Schema Mapping and ETL. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 1307–1316.
5. Fagin R. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In Complexity of Computation, SIAM-AMS Proceedings, Vol. 7, R.M. Karp (ed.), 1974, pp. 43–73.
6. Fagin R. Inverting Schema Mappings. *ACM Trans. Database Syst.*, 32(4):24, 2007.
7. Fagin R., Kolaitis P.G., Miller R.J., and Popa L. Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, 336(1):89–124, 2005.
8. Fagin R., Kolaitis P.G., Popa L., and Tan W.C. Composing Schema Mappings: Second-Order Dependencies to the Rescue. *ACM Trans. Database Syst.*, 30(4):994–1055, 2005.
9. Lenzerini M. Data Integration: A Theoretical Perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
10. Madhavan J. and Halevy A.Y. Composing Mappings Among Data Sources. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 572–583.
11. Maier D., Mendelzon A.O., and Sagiv Y. Testing Implications of Data Dependencies. *ACM Trans. Database Syst.*, 4(4):455–469, 1979.
12. Nash A., Bernstein P.A., and Melnik S. Composition of Mappings Given by Embedded Dependencies. *ACM Trans. Database Syst.*, 32(1):4, 2007.
13. Popa L., Velegrakis Y., Miller R.J., Hernández M.A., and Fagin R. Translating Web Data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 598–609.
14. Yu C. and Popa L. Semantic Adaptation of Schema Mappings when Schemas Evolve. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 1006–1017.

Schema Matching

ANASTASIOS KEMENTSIKIDIS
IBM T.J. Watson Research Center, Hawthorne,
New York, USA

Synonyms

Attribute or value correspondence

Definition

Schema matching is the problem of finding potential associations between elements (most often attributes or relations) of two schemas. Given two schemas S_1 and S_2 , a solution to the schema matching problem, called a *schema matching* (or more often a matching), is a set of *matches*. A match associates a schema element (or a set of schema elements) in S_1 to (a set of) schema elements in S_2 . Research in this area focuses primarily on the development of algorithms for the discovery of matchings. Existing algorithms are often distinguished by the information they use during this discovery. Common types of information used include the schema dictionaries and structures, the corresponding schema instances (if available), external tools like thesauri or ontologies, or combinations of these techniques. Matchings can be used as input to *schema mappings* algorithms, which discover the semantic relationship between two schemas.

Historical Background

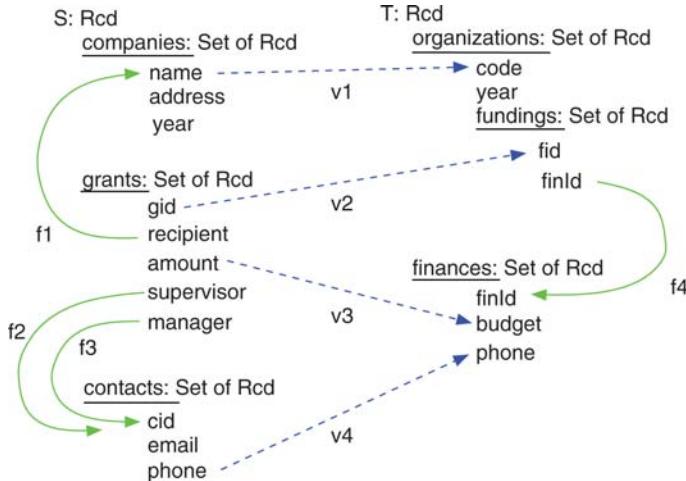
A *schema matching* is most often a binary relation between the elements of two schemas, but may, in a few approaches, be a relation between sets of elements in different schemas. In general, a matching represents a potential semantic relationship, but does not specify the semantics. For example, a matching between attributes $S_1.A$ and $S_2.B$ indicates that there may be some semantic relationship between these attributes. Examples of possible semantic relationships include subset relationships (e.g., all values of $S_1.A$ are also values of $S_2.B$) or has-a relationships (e.g., each value of $S_2.B$ has-a $S_1.A$ value). Consider the matches (v_1 through v_4) in Fig. 1. The matching helps in understanding the possible relationship between the schemas, but is not sufficient to determine how to transform data or queries from one schema to another. In contrast, a *schema mapping* (or mapping) is a specification of the semantic relationship between schemas [8]. The discovery

of matchings between elements of different schemas has been studied for decades, most notably in the context of the *schema integration* problem [1]. A solution to the schema integration problem presumes the ability to discover elements in the various schemas that are potentially *semantically related*, including those that may represent the same real-world concepts. Schema matching algorithms attempt to find candidate elements that may have a semantic relationship, though notably, they do not attempt to specify (or differentiate) the semantics of the relationship. These algorithms have been motivated by the presence of naming or structural differences (referred to as *conflicts*) among schemas that have been developed independently. Such differences are due to the fact that a real-world concept might have a different name or representation in different schemas. Schema integration deals with the development of methodologies to *discover* matchings in the presence of conflicts, but the main focus is on how each methodology *resolves* such conflicts (e.g., through schema transformations) so that the real-world concept is *uniquely* represented in the global schema. Indeed, from the five generic schema integration steps in each methodology, identified in [1], three of them deal with the resolution of conflicts. On the other hand, schema matching deals exclusively with the development of algorithms to *discover* matchings (see [9] for a survey of matching approaches).

Foundations

A matching between a pair of schemas S_1 and S_2 is typically a binary relation between the elements of the two schemas. In such cases, the so-called *local cardinality* of the matching is said to be one-to-one (1:1). Some algorithms consider matchings between sets of elements and, in the terminology of [9], are said to have a many-to-many local cardinality (see Figs. 2 and 3). In most approaches, the matching is between individual attributes of the schemas (Fig. 1). Matching algorithms compute a matching between S_1 and S_2 through a process which can abstractly be described by the following steps:

1. Consider (possibly all) pairs of elements s_1, s_2 with $s_1 \in S_1$ and $s_2 \in S_2$.
2. Compute a *score* indicating the confidence in the validity of each match between s_1 and s_2 .

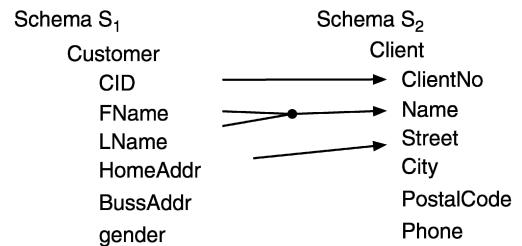


Schema Matching. Figure 1. A matching between source and target schemas.

3. Compute a matching by filtering and selecting a subset of the matching elements of the previous step.

Existing matching algorithms differ on how they implement each of these steps. Each implementation needs to make some key decisions in each step, hence the substantial diversity of existing solutions. During the first step, an important consideration is whether every possible pair of schema elements will be considered as a candidate for a match [6], or whether there is a more sophisticated mechanism in place to prune the potentially unrelated element pairs considered [3]. Many approaches assume that an element can be associated with at most one element in another schema (a restriction referred to as (1:1) global cardinality in [9]). For matching algorithms that only associate elements (not sets of elements), this means the resulting matching is a simple 1:1 relation over the schema elements.

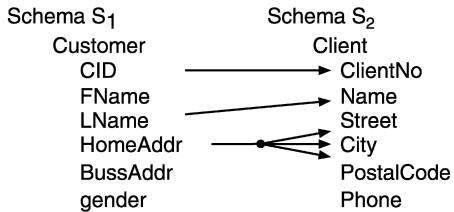
The second step is probably the most important, and there is a huge space of alternatives for the computation of scores. Score computation may take into account the name and types of schema elements, the structure of schemas and the corresponding nesting depths of elements, instance-level information (if instances of the schemas are available) like value ranges and patterns (for example, the frequency or position of substrings appearing in attribute values), or it may combine various types of information to compute a score. Matching algorithms are commonly classified by the type of information used during scoring computation (see the classification in [9]). The term *individual*



Schema Matching. Figure 2. A matching with n:1 local cardinality.

matcher is commonly used to describe algorithms that consider only a single (or a limited) type of information during score computation. Individual matchers are further classified into *schema-based* and *instance-based* depending on the type of information (i.e., schema versus instance) used during this phase. In contrast to individual matchers, *hybrid* or *composite* matchers rely on several types of information during score computation, where each type essentially corresponds to a different individual matcher. While hybrid matchers combine the results of multiple individual matchers in a *prespecified* manner, composite matchers are more flexible and allow for a dynamic composition of individual matchers which can be customized for the specific schemas being matched.

In the final step, there are two key considerations which influence the selection of matching elements that will comprise the result matching. First, the selection of matching elements is influenced by the supported *cardinality* which determines whether, or not, sets of



Schema Matching. **Figure 3.** A matching with $1:n$ local cardinality.

elements (for example, Street, City and PostalCode as a set as in Fig. 3) are considered in the matching.

The second consideration in this final step relates to how matching elements are selected based on their score. A common approach is to select matching elements whose scores are above a certain threshold and then select the matching elements with the maximum score, among the alternatives [6]. While such an approach results in matchings that are locally optimal, a more sophisticated approach considers maximizing the cumulative score of the matching elements in a matching [7].

State of the Art

There are many approaches to schema matching, so we offer a non-comprehensive overview of some of the representative approaches.

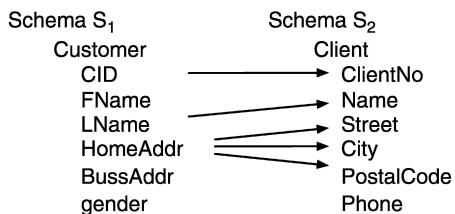
Cupid [6] is one of the first hybrid schema matching algorithms proposed in the context of model management. The algorithm considers initially every possible pair of elements in the two input schemas and thus its local cardinality is 1:1. It computes a linguistic and structural similarity score between these elements from which a weighted mean is computed using these two scores. The selection of matching elements in the resulting matching is performed by using a threshold over the computed scores and the supported global cardinality is 1:1, although it is suggested that matchings with global cardinality $1:n$ can also be supported (Fig. 4).

Coma [4] is a composite matcher with an extensible library of single and hybrid matchers. For example, Cupid might become one of the matchers used by Coma. Both the local and global cardinality is 1:1, and each *component* matcher of COMA computes a score between every pair of elements in the input schemas. Being a composite matcher, emphasis in Coma is given on how the results of component matchers are combined and four alternative strategies

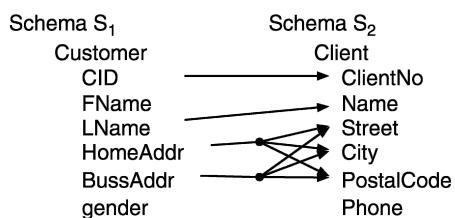
are proposed to this end. The four strategies compute the score of a matching element by taking the max, min, average or weighted sum of scores, computed for this element, of the component matchers. Experiments with these four strategies show that average gives the best results on the schemas tested.

The majority of matchers discover matchings whose local cardinality is 1:1. The iMap [3] matcher in contrast emphasizes the discovery of complex matching elements between two schemas, i.e., matchings with a local cardinality of $n:1$. For each element in the target schema, iMap employs a set of *specialized searchers* to discover candidate sets of elements in the source schema that together can be associated to the target schema element. Examples of the matchers supported are a numerical matcher, which discovers matches between elements containing numerical values; a categorical matcher, for categorical attributes; a unit conversion matcher; and a date matcher. In terms of global cardinality, iMap supports $n:1$ matchings (Fig. 5), since it allows an element to participate in more than one complex matching. An interesting feature of iMap is that apart from discovering (candidate) matchings, it also provides a module which traces the key decisions made by the system during matching discovery and it can therefore present the reasoning behind a suggested matching, in a human understandable format.

Kang and Naughton [5] make the interesting observation that matching algorithms often rely on *interpreting* the element names and values in the two input schemas, that is, they assume that the names used to described the same real-world concept or entity are syntactically and semantically related (e.g., a relational column named COLOR in S_1 versus one called PAINT in S_2). Therefore, when different element names are used, for the same elements in the two schemas (e.g., the former column is called CID in S_1 and PMS (PMS stands for the Pantone Color Matching System used in various industries.) in S_2), or different data encodings are used for the same real-world domain (e.g., different encodings for colors), then existing matching algorithms fail to discover appropriate matching elements. To this end, Kang and Naughton propose an instance-based matching algorithm that does not interpret values. Their matcher relies on the well-known notions of entropy and mutual-information, from Information Theory, to discover a matching between two input schemas. In a nutshell, their approach consists of two main steps. First, for each input



Schema Matching. Figure 4. A 1:1 local and 1:n global cardinality matching.



Schema Matching. Figure 5. A 1:n local and n:1 global cardinality matching.

schema, it computes the mutual information between each pair of attributes within the schema. The second step considers each possible matching with local and global cardinality of 1:1 and computes a score for this matching, a computation that takes into account the mutual information and entropy of the matched elements.

Key Applications

Schema Mapping, Schema Integration

Future Directions

Schema matchings represent potential associations between a pair of schema elements (or between two sets of schema elements). Matching algorithms do not discover the meaning of the association. A match between elements $S_1.A$ and $S_2.B$ may be discovered because these attributes contain similar values, because their names have a small edit-distance, their names are related in a domain ontology, or for numerous other reasons. Hence, matchings by themselves are not directly useful until they have been interpreted, either by a human, or a system designed to infer the semantic relationship between the elements. The most common example of the latter are schema mapping algorithms, which are designed to infer the semantic relationship between two schemas.

Cross-references

- ▶ Information Integration
- ▶ Metadata
- ▶ Schema Mapping

Recommended Reading

1. Batini C., Lenzerini M., and Navathe S.B. A comparative analysis of methodologies for database schema integration. ACM Comput. Surv., 18(4):323–364, 1986.
2. Bernstein P.A., Halevy A.Y., and Pottinger R.A. A vision for management of complex models. ACM SIGMOD Rec., 29(4):55–63, 2000.
3. Dhamankar R., Lee Y., Doan A., Halevy A., and Domingos P. iMap: Discovering complex semantic matches between database schemas. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
4. Do H. and Rahm E. Coma – a system for flexible combination of schema matching approaches. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 610–621.
5. Kang J. and Naughton J.F. On schema matching with opaque column names and data values. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 205–216.
6. Madhavan J., Bernstein P.A., and Rahm E. Generic schema matching with cupid. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 49–58.
7. Melnik S., Garcia-Molina H., and Rahm E. Similarity flooding: a versatile graph matching algorithm. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 117–128.
8. Miller R.J., Haas L.M., and Hernández M.A. Schema matching as query discovery. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 77–88.
9. Rahm E. and Bernstein P.A. A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350, 1994.

Schema Normalization

- ▶ Design for Data Quality

Schema Tuning

PHILIPPE BONNET¹, DENNIS SHASHA²

¹University of Copenhagen, Copenhagen, Denmark

²New York University, New York, NY, USA

Definition

Schema tuning is the activity of organizing a set of table designs in order to improve overall query and update performance.

Historical Background

Table design entails deciding which tables to implement and which attributes to put in those tables. Other sections of this encyclopedia (design theory, normalization theory) discuss a mathematical model of table design to eliminate redundancy. Sometimes however redundancy can be good for performance, so database tuners must consider the possibility of a principled incorporation of redundancy.

Foundations

Normalization tends to break up the attributes of an application into separate tables. Consider the normalized schema consisting of two tables:

Blog(blog_id, author_id, title, numreaders) and
Author(author_id, author_city).

If one frequently wants to associate blogs with the city of their authors, then this table design requires a join on author_id for each of these queries. A denormalized alternative is to add author_location to *Blog*, yielding *Blog*(blog_id, author_id, product, numreaders, author_city) and *Author*(author_id, author_city).

The *Author* table avoids anomalies such as the inability to store the location of an author whose blogs are perhaps temporarily offline.

Comparing these two schemas, one can see that the denormalized schema requires more space and more work on insertion of a blog. On the other hand, the denormalized schema is much better for finding the authors in a particular city.

The tradeoff of space plus insertion cost vs. improved speeds for certain queries is the characteristic one in deciding when to use a denormalized schema. Good practice suggests starting with a normalized schema and then denormalizing sparingly.

Redundant Tables

The previous example showed that redundancy can be helpful. The form of redundancy there was to repeat an association between two fields (in this case between authors and their address) for every blog.

One may also consider a completely redundant table.

For example:

Blog(blog_id, author_id, product, numreaders, author_city)

Author(author_id, author_city).

City_Agg(city, totalreaders)

This improves performance if one frequently wants to know the total readers per author city, but imposes an update time as well as a small space overhead. The trade-off is worthwhile in situations where many aggregate queries are issued and an exact answer is required.

Key Applications

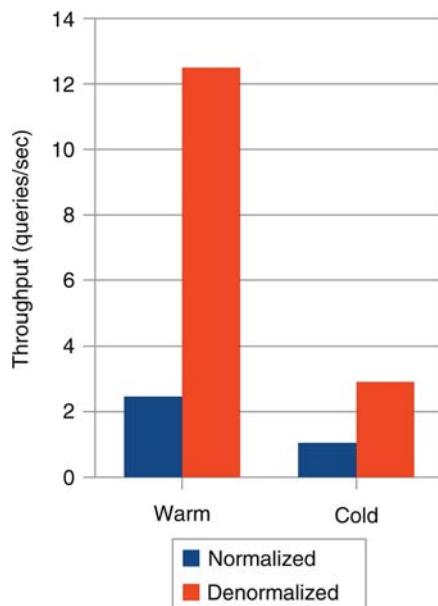
Schema tuning is relevant for all applications, but it is especially important for complex multi-table queries, particularly involving aggregates. Data warehousing applications typically include denormalized, redundant schemas, because data warehouses can be engineered to be updated at off hours and then intensively queried during the work day.

Experimental Results

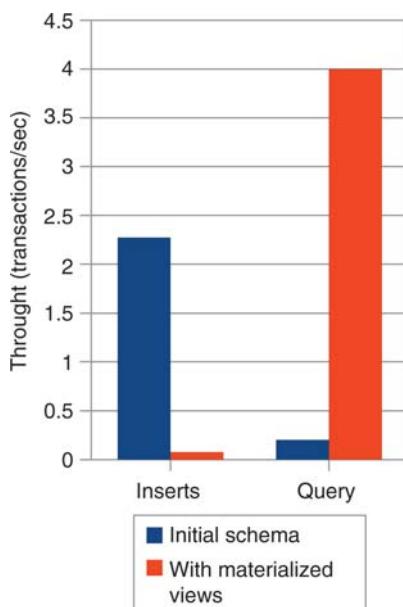
Denormalization

This experiment compares the performance impact of denormalization in the example presented above. Consider a query that finds the author in a given city. That query requires a join in the normalized schema whereas it requires a simple lookup in the denormalized schema.

Figure 1 presents the performance figures running this example on MySQL 6.0. The author table is



Schema Tuning. Figure 1. Denormalization experiment on MySQL 6.0.



Schema Tuning. Figure 2. Materialized view experiment on MySQL 6.0.

populated with 100,000 tuples, and the blog table with 50,000. Note that the denormalized schema provides a significant speed-off whether the cache is cold (in which case IOs are issued) or warm (the data already resides in the database cache).

Materialized Views

This experiment illustrates the trade-off between query speed-up and insert slow-down when using a materialized view. Consider the schema from the example above. The code includes a trigger in MySQL that maintains the materialized view when data are inserted in the blog table. The experiment includes the insertion throughput as well as query throughput. The query find the total number of readers per city.

Figure 2 shows the expected trade-off. Insertions are much slower with the materialized view due to trigger execution. Queries are much faster, however, because the query requires a join and an aggregate computation under the initial schema, whereas the query requires only a simple lookup when using the materialized view.

URL to Code and Data Sets

Denormalization experiment: <http://www.databasetuning.org/sec = denormalization>

Materialized view experiment: http://www.databasetuning.org/sec = materialized_views

Cross-references

- ▶ Application-level Tuning
- ▶ Clustering Index
- ▶ Design Theory
- ▶ Normalization Theory
- ▶ Performance Monitoring Tools

Recommended Reading

1. Celko J. and Joe Celko's. SQL for Smarties: Advanced SQL Programming (3rd Edn.). Morgan Kaufmann, San Francisco, CA, 2005.
2. Kimball R. and Ross M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (2nd Edn.). Wiley, New York, NY, 2002.
3. Shasha D. and Bonett P. Database Tuning: Principles, Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Francisco, CA, 2002.
4. Tow D. SQL Tuning. O'Reilly, North Sebastopol, CA, 2003.

Schema Versioning

JOHN F. RODDICK

Flinders University, Adelaide, SA, Australia

Definition

Schema versioning deals with the need to retain current data, and the ability to query and update it, through alternate database structures. (The structure of a database is held in a *schema* (pl. *schemata* or *schemas*). Commonly, particularly in temporal databases, these schemata represent the historical structure of a database but this may not always be the case.) *Schema Versioning* requires not only that data are not lost in schema transformation but also requires that all data are able to be queried, both retrospectively and prospectively, through user-definable version interfaces. *Partial schema versioning* is supported when data stored under any historical schema may be viewed through any other schema but may only be updated through one specified schema version – normally the current or active schema. (Schema evolution and schema versioning has been conflated in the literature with the two terms occasionally being used interchangeably.)

Historical Background

Multiple versions of a database schema may exist for a number of reasons. First, as a result of changes in system functionality and the external environment,

the structure of a database system might change over time but the historical shape of the database might need to be retained. Second, future versions might be created to develop and test later versions of a system. Third, more than one schema may be required in parallel to access the same data in a number of ways. Temporal databases, because of their requirement to maintain the context of historical information, are particularly affected by schema change.

The idea of schema versioning was introduced in the context of OODBs with a number of systems implementing techniques to handle multiple schema (such as Encore [12], Gemstone [7] and Orion [1]), including those that might be required for reasons other than simple historical succession. For example, parallel, alternate schema might be required to conceptualize an idea from a number of semantically consistent but different perspectives. In particular, polymorphism was suggested as a mechanism for providing some stability when faced with changing schema [6].

In order to maintain long-established concepts such as soundness and completeness, algebraic extensions have also been discussed [3]. More recently, schema versioning has also been considered in the context of spatio-temporal databases [9] and meta-data management [2,4].

Foundations

Schema versioning is closely related to the concepts of schema integration and data integration – all deal with the problems of accessing data through schema that were not used when the data were originally stored. However, the idea of maintaining multiple schemata, and allowing data to be accessed through them, raises a number of issues.

- What is the significance of a difference between two schema (or two databases) and therefore what is the informational cost of the change?
- What are the atomic operations of schema translation or transformation and what happens to the data during these operations?
- Are there any modelling techniques that can be used?
- Are there any other side-effects or opportunities (for instance in query language support)?

Types of Schema Evolution

As outlined elsewhere, four forms of schema evolution can be envisaged - attribute, domain, relation and key

evolution. Moreover, one change may involve more than one type of evolution, such as changing the domain of a key attribute and may also be reflected in the conceptual model of the system. Importantly for schema versioning, the inverse function for each of these must be considered. For example, when a schema (merely) evolves by vertically splitting a relation in two with data being suitably transformed, for schema versioning to be allowed, active transformation functions must be provided if the old schema is still to be utilized.

Practical and Theoretical Limits of Schema Versioning

It has been shown that in order to update data stored under two different schemata using the opposite schemata, they must have equivalent information capacity – all valid instances of some schema S_1 must be able to be stored under S_2 and vice-versa [5]. Specifically, $S_1 \equiv S_2$ if $I(S_1) \rightarrow I(S_2)$ is bijective where $I(S)$ is the set of all valid instances of S . This means that, in theory, *full* schema versioning across nonequivalent versions of a schema is unattainable and much research in the area adopts the weaker concept of *partial schema versioning* in which data stored under any historical schema may be viewed through any other schema but may only be updated through one specified schema version - normally the current or active schema.

However, in practice, many schema changes that expand or reduce the information capacity of a schema can be done without loss of information. This is the case, for example, for domains defined too large for any of the data, or for the creation of subclass relations from a single relation where the subclass type attribute already exists. It is a common practice, where there is some ambiguity in the requirements definition of a system, to allow for a larger schema capacity – some of which may never materialize and, as a result, changes to schemata to adhere to the data actually collected are not uncommon. For example, allowing for time and date when only date is recorded in practice.

Thus the limits for *practical schema versioning* in a database \mathfrak{D} are that $S_1 \xrightarrow{P} S_2$ (S_1 and S_2 have practical equivalent information capacity) if $I'(\mathfrak{D}|S_1) \rightarrow I'(\mathfrak{D}|S_2)$ is bijective where $I'(\mathfrak{D}|S_n)$ is the set of all instances of S_n inferrable from \mathfrak{D} given the constraints of S_n . This means that whether the integration of two schema is possible is dependent on the data held as well as the schema definition and while

this makes the ability to undertake wholesale change less predictable, it may provide an acceptable level of support in many practical situations.

Completed Schemas

In order to make all data for a relation available without the need to issue multiple queries, each targeting different time periods, the concept of a *completed schema*, C , can be employed that includes all attributes that have ever been defined over the life of a relation. The domain of each attribute in C is considered syntactically general enough to hold all data stored under every version of the relation and the implicit primary key of C is defined as the maximal set of key attributes for the relation over time. Depending on the mechanism used to implement schema versioning, the completed schema can then be used by a series of view functions. For example, in Fig. 1, V_{t_5} maps the completed schema C to a subset of the attributes in a schema S_{t_5} active during t_5 . A converse view function W_{t_2} maps from S_{t_2} to C .

Thus the data stored during t_2 may be mapped to the format specified during t_5 through invocation of $V_{t_5}(W_{t_2}(S_{t_2}))$.

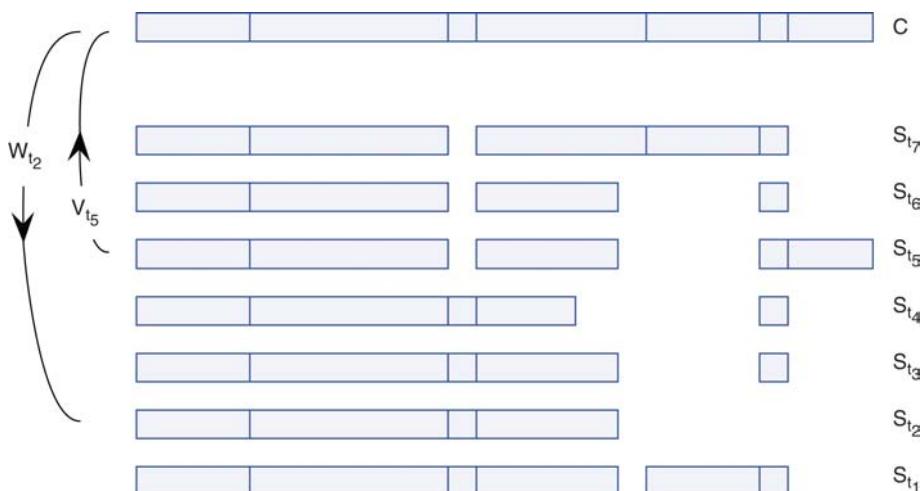
Query Language Support

Support for schema versioning does not yet exist in commercially available query languages. However, the TSQL2 proposal [10] and an earlier SQL/SE proposal [8] outlined some parts of the solution. As examples of such extensions:

- Reference to the *completed schema* can be included to provide access to all data;
- The specification of the schema could be done either through the specification of a global *schema-time* as in TSQL2, which would be useful for SQL embedded in a program with the schema-time set to compile time, or explicitly as part of the query;
- Attribute definition might be able to be tested by adding a test to see if a value was missing because it was *not defined* rather than being merely *null*;
- The language may also include meta-data queries such as the ability to ask what version of the schema a given piece of data adheres to.

Instance Amendment

For schema versioning in which the old schemata are still considered valuable, once a schema change is accepted, there are three options regarding the change to existing data. First, the underlying instances may be coerced to the new structure. While conceptually simple, this may result in lost information and an inability to reverse schema amendments. Secondly, data are retained in the format in which it was originally stored. This retains information content at the expense of more complex (and slower) translation of data when needed. Third, data are initially retained in the format in which it was originally stored but is converted when amended. While the most complex option, it has the advantage of identifying data that has not been amended since the schema change.



Schema Versioning. Figure 1. Versions of Schemata over time.

Future Directions

Schema versioning research has moved from low-level handling of syntactic elemental changes to more model-directed semantic handling of change. There are a number of other issues that make schema versioning non-trivial. Some of these represent future issues to be investigated.

- Many schema change requirements involve composite operations and thus a mechanism for schema level commit and rollback functions could be envisaged which could operate at a higher level to the data level commit and rollback operations.
- Access rights considerations are particularly a problem in object-oriented database systems. Consider, for example, a change to a class (e.g., Employees) from which attributes are inherited to a sub-class (e.g., Engineers) for which the modifying user has no legitimate access. Any change to the definition of attributes inherited from the superclass can be considered to violate the access rights of the subclass. Moreover, in some systems ownership of a class does not imply ownership of all instances of that class.
- In temporal databases the concept of vacuuming (q.v.) allows for the physical deletion of temporal data in cases where the utility of holding the data are outweighed by the cost of doing so [11]. Similar consideration must be given to the deletion of obsolete schema definitions, especially in cases where no data exists adhering to either that version (physically) or referring, through its transaction-time values, to the period in which the definition was active.

Cross-references

- ▶ Conceptual Modelling
- ▶ Schema Evolution
- ▶ Temporal Algebras
- ▶ Temporal Evolution
- ▶ Temporal Query Languages

Recommended Reading

1. Kim W., Ballou N., Chou H.T., Garza J.F., and Woelk D. Features of the orion object-oriented database system. In Object-Oriented Concepts, Databases and Applications. W. Kim and F. Lochovsky (eds.). ACM Press, New York, 1989, pp. 251–282.
2. Madhavan J. and Halevy A.Y. Composing mappings among data sources. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 572–583.

3. McKenzie L. and Snodgrass R. Schema evolution and the relational algebra. *Inf. Syst.*, 15(2):207–232, 1990.
4. Melnik S., Rahm E., and Bernstein P.A. Rondo: a programming platform for generic model management. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 193–204.
5. Miller R., Ioannidis Y., and Ramakrishnan R. The use of information capacity in schema integration and translation. In Proc. 29th Int. Conf. on Very Large Data Bases, 1993, pp. 120–133.
6. Osborn S. The role of polymorphism in schema evolution in an object-oriented database. *IEEE Trans. Know. Data Eng.*, 1(3):310–317, 1989.
7. Penney D. and Stein J. Class modification in the gemstone object-oriented DBMS. In Proc. 1987 Conf. on Object-Oriented Programming Systems, Languages, and Applications, 22(12): 111–117, 1987.
8. Roddick J.F. SQL/SE - a query language extension for databases supporting schema evolution. *ACM SIGMOD Rec.*, 21(3):10–16, 1992.
9. Roddick J.F., Grandi F., Mandreoli F., and Scalas M.R. Beyond schema versioning: a flexible model for spatio-temporal schema selection. *Geoinformatica*, 5(1):33–50, 2001.
10. Roddick J.F. and Snodgrass R. Schema versioning support, Chapter 22. In *The TSQL2 Temporal Query Language*. R. Snodgrass (ed.). Kluwer, Boston, 1995, pp. 427–449.
11. Skyt J., Jensen C.S., and Mark L. A foundation for vacuuming temporal databases. *Data Know. Eng.*, 44(1):1–29, 2003.
12. Zdonik S. Version management in an object-oriented database. In Proc. Int. Workshop on Adv. Programming Env., 1986, pp. 405–422.

Scientific Databases

AMARNATH GUPTA

University of California-San Diego, La Jolla, CA, USA

Definition

Scientific data refers to data that arise from scientific experiments, instruments, analytical tools, and computations. A chemistry experiment, for example, can yield data about the experimental setup, the pressure and temperature conditions under which the experiment was set up, measured variable like the heat released, initial and final masses the ingredients and products of the experiment, and so forth. The output of an instrument like a radio-telescope, after running signal processing algorithms, will produce “images” of the radio-frequency sources in a part of the sky that the telescope was looking at. A biologist, after obtaining the image of a dye-filled nerve cell, uses image analysis software to produce a set of measurements that reflect the structure of the cell and its subparts. Recently, environmental sensors are cast in oceans and send real-time

data on ocean temperature, salinity, oxygen content and other parameters. A scientific database refers to an information management framework need to store, organize, index, query, analyze, maintain, and mine such heterogeneous scientific data.

Historical Background

Investigation in data management techniques for scientific data started with studying file organization principles that are tuned toward specific kinds of scientific data [5], where the problem explored was to develop a multi-query indexing scheme for scientific records. Discipline-specific systems with data retrieval capabilities were also developed. Coughran [2] describes a system called Hydrosearch for worldwide hydrographic data from oceans that supported range queries like:

```
OCEAN = PACIFIC  
OUTPUT = REPORT  
LATHEM = N  
LONHEM = W  
LATDEG GEQ 31 AND LATDEG LEQ 33  
LONDEG GEQ 121 AND LONDEG LEQ 123  
MONTH GEQ 3 AND YEAR = 63 OR MONTH  
LEQ 2  
MXDPTH GEQ.98 DBOT
```

Around the same time, the National Laboratories dealing with a wide category of data related to energy research, recognized that “such diverse data applications as material compatibility, laser fusion, magnetic fusion, test, equation of state, weather, environmental and demographic data, has an acute need for a Scientific Data Base Management System (SDBMS)...The large volume of data, the numeric values within an epsilon of accuracy, the unknown data relationships, the changing requirements, coupled with the overall goal of extracting new intelligence from the raw data, dictate a data base system tailored toward scientific applications. Such an SDBMS should support scientific data types, a relational end user view, an interactive user language, interfaces to graphical and statistical packages, a programming language interface, interfaces to existing facilities, extensibility, portability, and use in a distributed environment” [1]. The system in [1] was developed on the CODASYL model [7].

It was in the 1980s that these different efforts started to take a coherent form where some general characteristics of scientific data management were identified. It was recognized through papers like [10]

that unlike business data that is usually defined by a data schema and values conforming to the schema, scientific data can come with a measurement framework, a metadata specification, and often a summarization framework. In [11] the database requirements for scientific data were characterized, and a new conference started in 1986 devoted to the issues managing scientific and statistical data.

Foundations

As alluded to in the previous paragraph, scientific data are usually more than the values of attributes – they are often accompanied by additional descriptors which together specify the semantics of the data and therefore determines how the data can be interpreted for query and analytical operations. Some broad categories of these additional components that specify the context of the raw scientific data are described below.

Measurement Framework

A measurement framework is a specification of the setting of the experimental data. For time-series data, it may be the sampling frequency. For spatial data represented as a raster, it includes the resolution of the grid, and how the measurement is obtained per grid (e.g., once at the center of the grid, average of n measurements within the grid...). For finite element data such as data from a fluid mechanics model, it may be the nature and regularity of the mesh over which data are recorded. The measurement framework is important in understanding the semantics of the data. If there are two raster data sets containing the measured temperature of two overlapping regions such that (i) one has a finer resolution than another, and (ii) one has the average-of-the-grid semantics, while the other has a single-sample-at-center semantics, how can one define a join operation to combine the two data sets? One cannot simply define a band join without considering a way to homogenize the data sets before they can be joined. Another aspect of the measurement framework is an assessment of the uncertainty associated with the measured, estimated or computed data. When data are associated with uncertainty, the traditional data models do not suffice – probabilistic or uncertainty-aware data models and query evaluation techniques are needed.

Metadata Framework

Metadata refers to descriptors that provide additional semantics beyond the value of an attribute. These

include the unit of measurement, the precision and accuracy of the data value, the uncertainty associated with the temporal or spatial position at which the data are taken, the experimental setting including whether the data are absolute or relative to any other reference, what if any computational corrections should be made on the data before it can be delivered to an end user. Metadata also covers constraint statements that limit the allowable domain of data values, or additional conditions that must be satisfied for the data to be interpreted. These may range from simple encoding schemes that specify “out of range” or “unknown” values, to multi-attribute constraints like “data are valid only if the cloud cover coefficient at the location is less than 0.2.” For data that are produced by computational algorithms (such as simulations of natural phenomena), the metadata also consists of the parameter settings of the algorithms, which must be taken into account to interpret, compare and analyze the data.

Summarization Framework

Scientific data are often voluminous due to high degree of sampling, or the total time or space over which data are acquired. In many applications, the total amount of data are too much and non-informative, and the scientists maintain only summarized versions of the data. For example, one may keep only weekly average temperature obtained from satellites; alternatively, one might keep the information only when there is a significant local change in the data. Since this is a common practice in many scientific disciplines, specification of how the data was summarized is a form of information that needs to go together with the data itself.

Heterogeneity of Types

An important characteristic of scientific data is the wide range of complexity and heterogeneity of the data types that are needed to model the applications.

Complexity and Heterogeneity of Formats

Distinct from the issue of data types, scientific data demonstrates a wide variety of formats for the same kind of data. In some domains, there is a lack of a single standard, and vendors of instruments that provide data, or vendors of software that manipulate the data define their own formats to facilitate their respective needs for data generation, analysis and

visualization. For example, biological pathways, which are essentially graphs with node and edge attributes are represented differently by software such as Cytoscape [9], PATIKA [3], Pathway Studio [6] and the standardization effort called BioPAX (www.biopax.org) In other domains, there is more standardization. But the formats are very complex. HDF (Hierarchical Data Format) is a complex scientific data format for storing multidimensional data, raster data and tables. It is also designed to be self-describing and contains additional metadata. The multiplicity of supported models and the embedded metadata requires special data management tools [4] to be developed for indexing and querying HDF data. A consequence of this format heterogeneity and complexity is that interoperability of scientific data remains a research challenge.

Data Management Issues in Scientific Databases

Traditional Issues In 1985, [11] identified a number of data management issues that pertain to scientific data management. Many of these issues that hold equally well today are:

- *Data Volume and Compression:* Much of scientific data are multidimensional. While the data sets can be very large, the fraction of the multidimensional space that is occupied by the data is smaller. This brings up the need to compress the data as well as to choose a data organization that will exploit the sparsity of data. Further, data manipulation and query evaluation techniques that utilize the compressed data or a new data organization are needed. Managing large-scale scientific data is now considered to be an important challenge. A notable project in this area is led by the Stanford Linear Accelerator (SLAC – <http://www.slac.stanford.edu/>) that is attempting to build a data management system for a petabyte of data.
- *Data Structures:* With new scientific data types, there is a need for new data structures and access methods. In recent years, a number of index structures have been proposed for multidimensional data. For example, Zhang et al. [12] proposed a data structure for sampling multidimensional data; Rotem et al. [8] have developed bitmap indexes for very large-scale multidimensional data.
- *New Operations:* Data manipulation and search in scientific databases need operators that go beyond

traditional relational or tree manipulation algebras. It requires new operations such as sampling, neighborhood searching in metric space, estimation and interpolation operators for sampled data over a dense data space, novel join methods for complex data types.

- **Analysis Support:** The ultimate goal of scientific data acquisition and storage is some form of analysis and derivation of scientific truth. Scientists, the primary users of the database, are not often willing to learn complex query languages – instead, they want to query the data as part of their analytical tasks. The management of the entire analysis process require analysis-friendly user interfaces that are sufficiently expressive but not overly complex, a way to facilitate repeated use of the same query with differing parameters, management of long-running queries, handling large volumes of intermediate data, and optimal execution of an entire analytical workflow.
- **Quality Management:** Quality awareness of data is important when the data collected in a database comes from any error prone process. For scientific data, errors and approximations arise often due to factors like resolution limits of instruments, malfunctioning of devices, unforeseen environmental or experimental confounding factors, biases introduced by sampling, approximations used by pre-processing computations and, of course, human error. The problem gets compounded when a data product is derived from an existing data product. In many cases, data for a given application may come from data sources with different quality and “believability.” Query languages, evaluation techniques and analysis for scientific data needs to be quality-aware, and give a user the ability to filter data based on its quality and integrity.

Recent Trends More recently, the scientific data management community has identified newer challenges over and above the issues above. Some of them are:

- **Annotation Management:** Annotating data is a common practice in science. An annotation is a piece of user-imposed data that references an arbitrary data element in an existing data store. One can annotate a block of data with a statement about its quality; one may annotate a fragment of data with information of its provenance (i.e., where the data was obtained from and how it was transformed before

it appeared in its present form); one may annotate data by tagging it with keywords or terms from an ontology so that it can be easily related to other data. Annotation management attempts to create a uniform way to store the annotation and their referent data so that both the primary data and the annotations can be queried together.

- **Semantics:** In many domain sciences, the semantics of data is not adequately represented in data repositories. This makes it very difficult to one user to interpret data from another user, and even harder to combine multiple kinds of data together. This recognition has led to a renewed interest in developing semantic data models for scientific data. As part of this effort ontologies are being created to standardize and define the terms and the inter-term relationships in a discipline using standards like the Web Ontology Language, OWL, so that data producers can either use the ontological terms to represent their data, or map their existing data to the ontologies. At the same time, efforts are underway to develop query, integration and data mining techniques that make use of the semantic framework.

Cross-references

- ▶ [Annotation Management](#)
- ▶ [Data Types in Scientific Data Management](#)

Recommended Reading

1. Birss E.W., Jones S.E., Ries D.R., and Yeh J.W. Scientific data base management at Lawrence Livermore Laboratory: needs and a prototype system. Technical Report UCRL-80146; CONF-771062-1, Lawrence Livermore Lab, California University, 1977.
2. Coughran E. HYDROSEARCH, an easy-to-use retrieval system for hydrographic station data. OCEANS, 7:418–421, 1975.
3. Demir E., Babur O., Dogrusoz U., Gursoy A., Nisanci G., Cetin-Atalay R., and Ozturk M. PATIKA: an integrated visual environment for collaborative construction and analysis of cellular pathways. Bioinformatics, 18(7):996–1003, 2002.
4. Gosink L., Shalf J., Stockinger K., Wu K., and Bethel W. HDF5-FastQuery: accelerating complex queries on HDF datasets using fast bitmap indices. In Proc. 18th Int. Conf. on Scientific and Statistical Database Management, 2006, pp. 149–158.
5. Ikeda H. and Naito M. Evaluation of a combinatorial file organization scheme of order one. In Proc Study on Scientific Database Management Systems, 1979, pp. 195–199 (in Japanese).
6. Nikitin A., Egorov S., Daraselia N., and Mazo I. Pathway studio – the analysis and navigation of molecular networks. Bioinformatics, 19(16):2155–7, 2003.

7. Olle T.W. *The Codasyl Approach to Data Base Management*. Wiley, Chichester, UK, 1978.
8. Rotem D., Stockinger K., and Wu K. Minimizing I/O costs of multi-dimensional queries with bitmap indices. In Proc. 18th Int. Conf. on Scientific and Statistical Database Management, 2006, pp. 33–44.
9. Shannon P., Markiel A., Ozier O., Baliga N.S., Wang J.T., Ramage D., Amin N., Schwikowski B., and Ideker T. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Res.*, 13(11):2498–2504, 2003.
10. Shoshani A., Olken F., and Wong H.K.T. Characteristics of scientific databases. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 147–160.
11. Shoshani A. and Wong H.K.T. Statistical and scientific database issues. *IEEE Trans. Softw. Eng.*, 11(10):1040–1047, 1985.
12. Zhang X., Kurc T., Saltz J., and Parthasarathy S. Design and analysis of a multi-dimensional data sampling service for large scale data analysis applications. In Proc. 20th Int. Parallel and Distributed Processing Symp., 2006.

Scientific Knowledge Bases

- Biomedical Scientific Textual Data Types and Processing

Scientific Medicine

- Evidence Based Medicine

Scientific Query Languages

- Query Languages for the Life Sciences

Scientific Visualization

RONALD PEIKERT
ETH Zurich, Zurich, Switzerland

Definition

Scientific visualization [1] provides graphical representations of numerical data for their qualitative and quantitative analysis. In contrast to a fully automatic analysis (e.g., with statistical methods), the final analytic step is left to the user, thus utilizing the power of the human visual system. Scientific visualization differs

from the related field of information visualization in that it focuses on data that represent samples of continuous functions of space and time, as opposed to data that are inherently discrete.

The challenge in scientific visualization is to cope with massive data, which cannot be presented to the user in an unprocessed way for several reasons:

1. Volumetric data, i.e., data given on a three-dimensional domain, occlude each other. This problem becomes even more severe if data are not scalars, but vectors or even tensors.
2. Visualization should provide a global picture of the spatial and temporal behavior of the data, but also allow for interactive exploration of details.
3. There can be multiple data (different physical quantities, multiple data channels, etc.) at each point in the domain.
4. Visualization of scientific data should also include visualization of their uncertainty.
5. The amount of raw data often exceeds limitations of processor speed, transfer rates, memory size, and display resolution.

Applications of scientific visualization cover a wide spectrum of science and engineering disciplines. Currently, some of the most active fields are medical and biomedical image data, simulation and measurement data from fluid or solid mechanics, molecular data, data from geology and geophysics, astronomy, weather and climate.

Key Points

Scientific visualization evolved in the 1980s from earlier graphing techniques when 3D computer graphics opened new ways of displaying numerical data. The abstraction from the application domain led to interdisciplinary visualization software systems with a modular dataflow architecture. This approach is still successful [2], since for many visualization tasks, the semantics of the data is less relevant than mathematical properties such as the discretization type or the categorization into scalars, vectors, and tensors.

For volumetric scalar data, important visualization techniques are isosurfaces and direct volume rendering. For vector fields examples are arrow glyphs, integral lines (streamlines, streaklines) and texture advection. Tensor fields are visualized with glyphs (ellipsoids, superquadrics) or tensor lines. In the special case of diffusion tensor MRI data, fiber tracking techniques are applied. Scalar,

vector, and tensor fields all are amenable to topology-based visualization, which provides both the singularities and a segmentation of the domain into regions of “similar data behavior.” More general, feature extraction and feature tracking techniques aim at reducing the data complexity and providing the viewer with only the most salient information. Features (e.g., edges, ridges, flow structures) are typically defined in terms of data and their derivatives. User-defined feature definitions are possible in visualization systems built on the linked views paradigm, where simultaneous views of both physical and data space are available all of which allow for interactions such as data coloring and subsetting. The visualization of very large data requires optimization techniques including multi-resolution, parallel and out-of-core algorithms, as well as view-dependent visualization.

Cross-references

- ▶ [Data Visualization](#)
- ▶ [Visualization Pipeline](#)

Recommended Reading

1. Hansen C.D. and Johnson C.R. (eds.). *Visualization Handbook*. Academic Press, San Diego, CA, 2004.
2. Schroeder W., Martin K., and Lorensen B. *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics* Kitware, Inc., New York, 2006.

Scientific Workflows

BERTRAM LUDÄSCHER, SHAWN BOWERS,
TIMOTHY MCPHILLIPS
University of California-Davis, Davis, CA, USA

Synonyms

[In silico experiment](#); [Grid workflow](#)

Definition

A *scientific workflow* is the description of a process for accomplishing a scientific objective, usually expressed in terms of *tasks* and their *dependencies*. Typically, scientific workflow tasks are computational steps for scientific simulations or data analysis steps. Common elements or stages in scientific workflows are acquisition, integration, reduction, visualization, and publication (e.g., in a shared database) of scientific data. The tasks of a scientific workflow are organized (at

design time) and orchestrated (at runtime) according to dataflow and possibly other dependencies as specified by the workflow designer. Workflows can be designed visually, e.g., using block diagrams, or textually using a domain-specific language.

Historical Background

Workflows have a long history in the database community and in business process modeling, in which case they are sometimes called *business workflows* to distinguish them from scientific workflows. The database community realized early [10] that scientific data management has different characteristics from more traditional business data management. Early work on scientific workflows within the database community took a database-centric view by defining data models and query languages suitable for scientific experiment management systems. The MOOSE data model and FOX query language have their roots in the late 1980’s [5] and early 1990’s [13] and gave rise to the ZOO experiment management environment [6], an early system based on an underlying object-oriented database. Another pioneering work that emphasized the importance of workflow concepts in scientific data management is WASA, a *Workflow-based Architecture for Scientific Applications* [8]; the related publication [12] introduced the term “scientific workflow” and contrasted such workflows with office automation and business workflows. An early benchmark comparing different database architectures for scientific workflow applications is LabFlow-1 [1].

Other roots of scientific workflow systems include *problem solving environments*, which emerged in the nineties in the computational sciences community as intuitive tools to “solve a target class of problems for scientific computing” [4], and *laboratory information management systems* (LIMS) [9], which can be seen as special scientific workflow systems that are used in a laboratory environment for the management of samples, instrument-based measurements, and other functions, including data analysis and workflow automation. Similar to many scientific workflow systems, problem solving environments and LIMS sometimes employ a visual programming paradigm to link together components. An early, if not the first, visual language that allowed simple interfacing with lab instruments was G in LabVIEW1.0, released in 1986 for the Apple Macintosh. Modern incarnations of LIMS can include functions of enterprise resource planning (ERP) systems

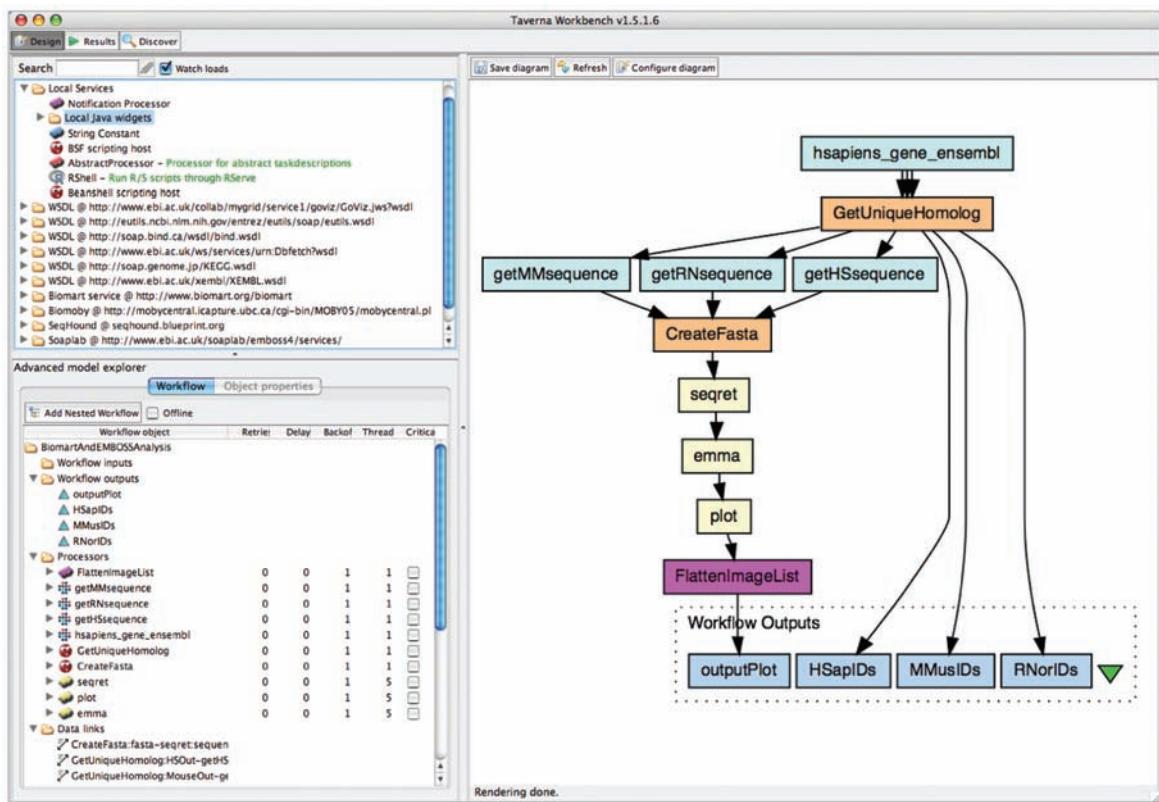
and thus go beyond the scope of current scientific workflow systems.

With the advent of *e-Science* as a paradigm, scientific workflow research and development has seen a major resurgence. Similar to the related term *cyberinfrastructure*, e-Science brings together computational techniques and tools from the computational sciences, distributed and high-performance computing, databases, data analysis, visualization, sharing, and collaboration. There are now a number of new open source as well as commercial scientific workflow systems available and under active development. For example, a special journal issue of *Concurrency and Computation: Practice and Experience* covers a number of systems, including Kepler, Taverna, and Triana among others [2]. For a high-level overview and attempt at a classification of current scientific workflow systems see [14],

which includes also references to many other systems, such as Askalon, Pegasus/DAGMan, Karajan, etc.

Foundations

Science is an exploratory process involving cycles of observation, hypothesis formation, experiment design and execution. Today, scientific knowledge discovery is increasingly driven by data analysis and computational methods, e.g., due to ever more powerful instruments for observation and the use of commodity clusters for high-performance scientific computing and simulations in the computational sciences. Scientific workflows can be applied during various phases of the larger science process, specifically modeling and automation of computational experiments, data analysis, and data management. The results from workflow runs can yield new data and insights and thus may lead to affirmation,



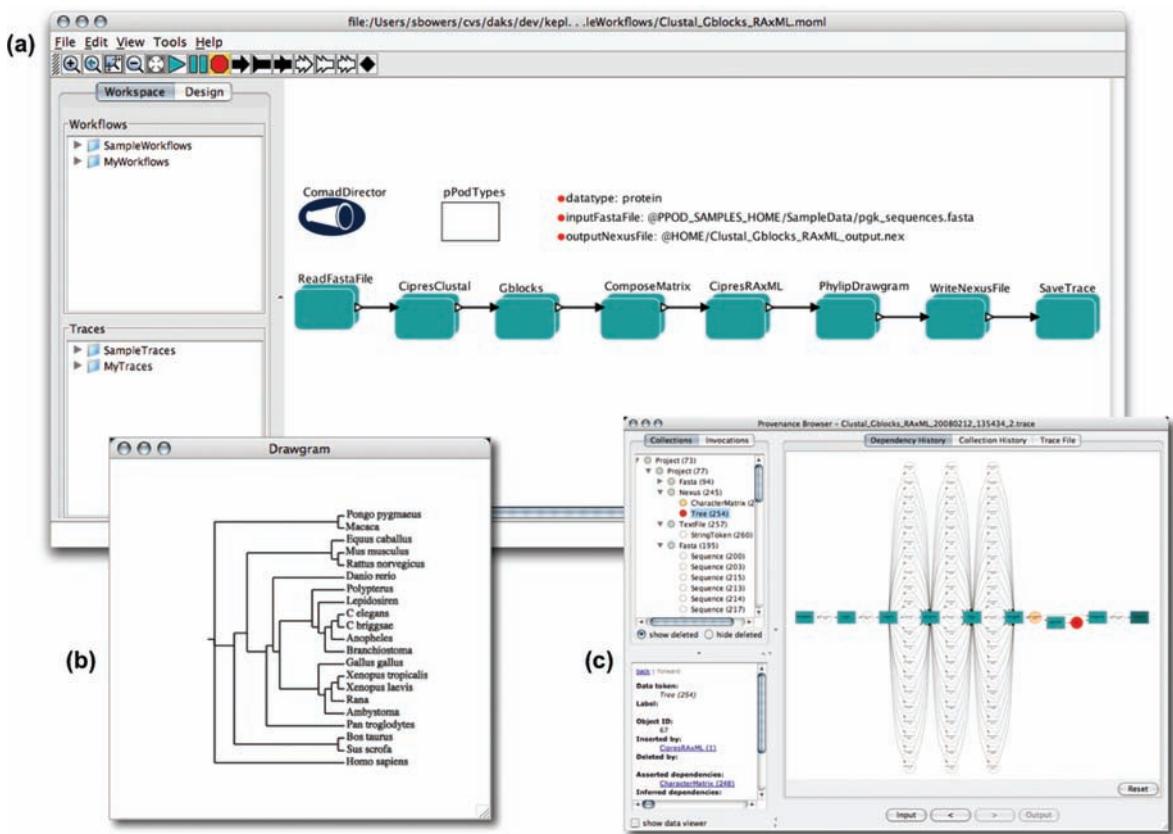
Scientific Workflows. Figure 1. Example workflow represented in the Taverna workflow system. This workflow extracts gene IDs from human chromosome 22 with mappings to disease functions and homologues in mouse and rat; fetches base pairs of the associated DNA sequences; combines the sequences into a FASTA file; performs a multiple sequence alignment; and renders the result. The workflow uses three soaplab-based analysis operations (seqret, emma, plot) that run on the EBI compute cluster.

modification, or refutation of a given hypothesis or experiment outcome.

Scientific workflow systems automate the execution of scientific workflows, and may additionally assist in workflow design, composition, and the management and sharing of workflow descriptions. Other important functions include support for workflow execution monitoring, for recording and querying provenance information, for workflow optimization (e.g., exploiting dataflow and concurrency information for parallel execution), and for fault-tolerant execution. These additional features also distinguish a scientific workflow systems approach from more traditional script-based solutions in which such functionality is usually not provided. Workflow provenance information can be used, e.g., to facilitate the interpretation, debugging, and reproducibility of scientific analyses. An increasing

number of scientific workflow systems now offer support for various forms of provenance. One can distinguish *data provenance*, i.e., the processing history of data, and provenance information describing the *workflow evolution*, i.e., the history of changes of a workflow definition and the parameter settings used for a particular workflow instance.

Scientific workflows are often visually represented as directed graphs (Figs. 1 and 2) linking atomic tasks or composite components, so-called *subworkflows*. Tasks can include native functions of the workflow system, but often correspond to invocations of local applications, remote (web) services, or subworkflows. Scientific workflows differ from conventional programming in that the workflows are often more coarse-grained and involve wiring together of pre-existing components and specialized algorithms. Figure 1



Scientific Workflows. Figure 2. Example scientific workflow in the Kepler system: (a) user interface for creating, editing, and executing scientific workflows; (b) a visual representation of the data product (a phylogenetic tree) computed by a workflow run; and (c) a viewer for navigating the data provenance (lineage) captured in an execution trace. This workflow uses a combination of local and remote (web) services to perform multiple sequence alignment and phylogenetic tree inference on input DNA sequences.

shows a simple bioinformatics workflow in the Taverna system, consisting of multiple (soaplab) services.

There is currently no standard *scientific workflow language*, and standards from related communities (e.g., BPEL4WS) have not found widespread adoption in the scientific workflow community. For example, job-based *grid workflows* are often represented as directed acyclic graphs (DAGs), which are then scheduled on a computational grid or cluster computer according to the implied task dependencies. In this *model of computation*, each task is executed only once per workflow run and task scheduling amounts to finding a *topological sort* for the partial order implied by the DAG. Other more sophisticated models of computation consider tasks as independent and continuously executing processes which can receive and send many different data items per workflow run. Scientific workflow systems that support such models of computation may thus be used for *data stream processing* and *continuous queries*. Similar to business workflows, formal approaches such as *Petri nets* can be used to describe scientific workflow execution semantics. However, the dataflow models of computation of many scientific workflow systems can exhibit both task- and pipeline-parallelism where token order is important. A standard computation model for such dataflow systems is the *Kahn Process Network* model. The structurally simple linear Kepler workflow in Fig. 2 is achieved via a special model of execution, implemented by a so-called *director*. (Kepler inherits from the underlying Ptolemy II system the capability to use distinct directors at different workflow modeling levels and thus to combine different models of computation in a single workflow.) The COMAD (*Collection-Oriented Modeling And Design*) director in Fig. 2 specifies that workflow components work on a continuous, XML-like data stream which passes through all components eventually. Each component is configurable to compute only on certain (tagged) data collections. Results are injected back into the stream. The resulting more linear workflows are easy to comprehend and evolve over time, another important advantage over script-based solutions.

Key Applications

Scientific workflows now span virtually all areas of the natural sciences. Bioinformatics is a particularly active application area (cf. Figs. 1 and 2), but the spectrum of disciplines employing scientific workflow systems is much wider and includes particle physics, chemistry,

neurosciences, ecology, geosciences, oceanography, atmospheric sciences, astronomy and cosmology, among others.

URL to Code

A number of open source scientific workflow systems are available, among them:

Kepler: <http://www.kepler-project.org>

Taverna: <http://taverna.sourceforge.net>

Triana: <http://www.trianacode.org>

For a list including many other systems, see <http://www.extreme.indiana.edu/swf-survey/>.

Cross-references

- ▶ [Business Process Modeling](#)
- ▶ [\(Business\) Workflow](#)
- ▶ [Data Analysis](#)
- ▶ [Dataflow](#)
- ▶ [Problem Solving Environment](#)
- ▶ [Provenance](#)
- ▶ [Visual Programming](#)

Recommended Reading

1. Bonner A.J., Shrufi A., and Rozen S. LabFlow-1: a database benchmark for high-throughput workflow management. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996, pp. 463–478,
2. Fox G.C. and Gannon D. (eds.) Concurrency and Computation: Practice and Experience. Special Issue: Workflow in Grid Systems, 18(10), 2006.
3. Gil Y., Deelman W., Ellisman W., Fahringer T., Fox G., Gannon D., Goble C., Livny M., Moreau L., and Myers J. Examining the challenges of scientific workflows. Computer, 40(12): 24–32, 2007.
4. Houstis E., Gallopoulos E., Bramley R., and Rice J. Problem-solving environments for computational science. IEEE Comput. Sci. Eng., 4(3):18–21, 1997.
5. Ioannidis Y.E. and Livny M. MOOSE: modeling objects in a simulation environment. In IFIP Congress, G.X. Ritter (ed.). North-Holland, 1989, pp. 821–826.
6. Ioannidis Y.E., Livny M., Gupta S., and Ponnekanti N. ZOO: a desktop experiment management environment. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 274–285.
7. Ludäscher B. and Goble C. (eds.) ACM SIGMOD Rec., 34(3): 44–49, September 2005. Special Issue on Scientific Workflows.
8. Medeiros C.B., Vossen G., and Weske M. WASA: a workflow-based architecture to support scientific database applications. In Proc. 6th Int. Conf. Database and Expert Syst. Appl., 1995, pp. 574–583.
9. Nakagawa A.S. LIMS: Implementation and Management. The Royal Society of Chemistry, Thomas Graham House. The Science Park Cambridge CB4 4WF, 1994.

10. Shoshani A., Olken F., and Wong H.K.T. Characteristics of scientific databases. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 147–160.
11. Taylor I., Deelman E., Gannon D., and Shields M. (eds.) Workflows for e-Science: Scientific Workflows for Grids. Springer, Berlin, 2007.
12. Wainer J., Weske M., Vossen G., and Medeiros C.B. Scientific workflow systems. In Proc. NSF Workshop on Workflow and Process Automation in Information Systems: State of the Art and Future Directions, 1996.
13. Wiener J.L. and Ioannidis Y.E. A moose and a fox can aid scientists with data management problems. In Proc. Fourth Int. Workshop on Database Programming Languages, 1993, pp. 376–398.
14. Yu J. and Buyya R. A taxonomy of scientific workflow systems for grid computing. ACM SIGMOD Rec., 34(3):44–49, September 2005. Special Issue on Scientific Workflows.

Key Points

Traditionally, screen scrapers have been used to interface legacy systems residing on old mainframes, which often host critical data processing applications. Although both hardware and software are obsolete, they cannot be replaced for various reasons. Screen scraping offers a cost-effective alternative to access and leverage underlying data stores. Typical applications include capturing emulated IBM 3270 screens (a widely used text-based protocol for dumb terminals). Combined with macros to enable navigation throughout different screens, scrapers can be used to integrate with modern architectures.

Common text scraping methods make heavy use of syntactic tools such as regular expressions to identify relevant data. Recently, more semantic approaches have been researched that furthermore allow scraping from unstructured documents such as PDF using generic document understanding techniques supplemented by domain-specific knowledge modeled with ontologies. Layout and table recognition can be performed on a visual level using top-down segmentation (recursive X-Y cut), bottom-up clustering as well as probabilistic graph-matching algorithms [1]. Identified document segments can then be classified using semantically designed rules in order to annotate the original document with its implicit structure.

Another key area is web scraping, which locates data by exploiting the explicit underlying layout markup (HTML) of its presentation. As a means to build interfaces (APIs) for web sites not available otherwise, scrapers also serve as the basis for state-of-the art semantic web applications called web mashups, such as MIT’s SIMILE project [2]. Web scrapers filter relevant content, serializing it in annotated XML format. The main complexity issue arising with all scraper types is coping with change: scrapers are said to “break,” when data presentation changes substantially. Visual IDEs can assist scraper design, offering lower maintenance effort compared to purely programmatical solutions.

Score Propagation

- ▶ Propagation-Based Structured Text Retrieval

Screen Scraper

HARALD NAUMANN

Vienna University of Technology, Vienna, Austria

Synonyms

Screen scraping; Data extraction; Screen wrapper

Definition

A screen scraper is a program which extracts relevant data from the visual user interface of an application. Input data are commonly represented using text-only or graphically enhanced tables, lists and forms, tailored to a human audience. Scraping is the task of collecting data from its presentation, not directly from its source for lack of access. The scraper output has a structured and machine-readable format, where extracted data are usually annotated with its semantics (metadata), suitable for automatic post-processing. The process can be thought of as reverse-engineering a data store from its presentation, abstracting content from layout. Using this approach, application data are taken from the human-oriented screen output rather than the application’s hidden proprietary data structures.

Cross-references

- ▶ Information Extraction
- ▶ Web Data Extraction
- ▶ Wrapper Generator
- ▶ Wrapper Maintenance

Recommended Reading

1. Hassan T. and Baumgartner R. Intelligent text extraction from PDF documents. In Proc. Int. Conf. on Intelligent Agents, Web Technologies and Internet Commerce, 2005, pp. 2–6.
2. Huynh D., Mazzocchi S., and Karger D. Piggy bank: experience the semantic web inside your web browser. In the Fourth Int. Semantic Web Conf., 2005.

Screen Scraping

- ▶ Languages for Web Data Extraction
- ▶ Screen Scraper

Screen Wrapper

- ▶ Screen Scraper

SCSI Target

KALADHAR VORUGANTI

Network Appliance, Sunnyvale, CA, USA

Definition

In SCSI protocol, the server which provides the storage is known as target. There can be multiple targets in a storage controller. Each target can offer access to either a single volume or multiple volumes. The volumes being offered by a storage target are mapped into LUNs by the host operating system.

Key Points

Storage controllers, JBOD (just a bunch of disks in an enclosure), direct attached disks, and storage virtualization boxes can all act as SCSI targets. Other types of storage media that can support the SCSI protocol can also act as SCSI targets. The transport protocol encapsulating the SCSI commands dictate the uniqueness of the SCSI target and initiator identifiers.

Cross-references

- ▶ LUN
- ▶ LUN Mapping
- ▶ Storage Protocols
- ▶ Volume

SDI, Selective Dissemination of Information

- ▶ Information Filtering

Search Advertising

- ▶ Web Advertising

Search Engine Caching and Prefetching

- ▶ Web Search Result Caching and Prefetching

SDC Score

JOSEP DOMINGO-FERRER

Universitat Rovira i Virgili, Tarragona, Catalonia

Definition

Statistical disclosure control (SDC) methods for microdata can be ranked based on information loss, disclosure risk or a combination of both. An SDC score is a combination of information loss and disclosure risk measures used to rank methods.

Key Points

The construction of an SDC score combining information loss and disclosure risk was first proposed in [1,2]. For each method M and parameterization P , the following score is computed:

$$\text{Score}(\mathbf{V}, \mathbf{V}') = \frac{IL(\mathbf{V}, \mathbf{V}') + DR(\mathbf{V}, \mathbf{V}')}{2}$$

where IL is an information loss measure, DR is a disclosure risk measure and \mathbf{V}' is the protected dataset obtained after applying method M with parameterization P to an original dataset \mathbf{V} .

In the above references, IL and DR were computed using a weighted combination of several information

loss and disclosure risk measures. With the resulting score, a ranking of a set of masking methods (and their parameterizations) was obtained. Yancey et al. [3] later followed the same approach to rank a different set of methods using a slightly different score.

To illustrate how a score can be constructed, the particular score used by [2] is next described. Let X and X' be matrices representing original and protected datasets, respectively, where all attributes are numerical. Let V and R be the covariance matrix and the correlation matrix of X , respectively; let \bar{X} be the vector of attribute averages for X and let S be the diagonal of V . Define V' , R' , \bar{X}' , and S' analogously from X' . The Information Loss (IL) is computed by averaging the mean variations of $X - X'$, $\bar{X} - \bar{X}'$, $V - V'$, $S - S'$, and the mean absolute error of $R - R'$ and multiplying the resulting average by 100. Thus, the following expression is obtained for information loss:

$$\begin{aligned} IL = & \frac{100}{5} \left(\frac{\sum_{j=1}^p \sum_{i=1}^n \frac{|x_{ij} - x'_{ij}|}{|x_{ij}|}}{np} + \frac{\sum_{j=1}^p \frac{|\bar{x}_j - \bar{x}'_j|}{|\bar{x}_j|}}{p} \right. \\ & + \frac{\sum_{j=1}^p \sum_{1 \leq i \leq j} \frac{|v_{ij} - v'_{ij}|}{|v_{ij}|}}{\frac{p(p+1)}{2}} + \frac{\sum_{j=1}^p \frac{|v_{jj} - v'_{jj}|}{|v_{jj}|}}{p} \\ & \left. + \frac{\sum_{j=1}^p \sum_{1 \leq i \leq j} |r_{ij} - r'_{ij}|}{\frac{p(p-1)}{2}} \right) \end{aligned}$$

The expression of the overall score is obtained by combining information loss and information risk as follows:

$$Score = \frac{IL + \frac{(0.5DLD + 0.5PLD) + ID}{2}}{2}$$

Here, DLD (Distance Linkage Disclosure risk) is the percentage of correctly linked records using distance-based record linkage, PLD (Probabilistic Linkage Record Disclosure risk) is the percentage of correctly linked records using probabilistic linkage, ID (Interval Disclosure) is the percentage of original records falling in the intervals around their corresponding masked values and IL is the information loss measure defined above.

Based on the above score, it turned out that, for the benchmark datasets and the intruder's external information they used in [2], two good performers among the set of methods and parameterizations they tried were: (i) rankswapping with parameter p around

15; (ii) multivariate microaggregation on unprojected data taking groups of three attributes at a time.

Cross-references

- ▶ Disclosure Risk
- ▶ Inference Control in Statistical Databases
- ▶ Information Loss Measures
- ▶ Microaggregation
- ▶ Microdata
- ▶ Rank Swapping
- ▶ Record Matching

Recommended Reading

1. Domingo-Ferrer J., Mateo-Sanz J.M., and Torra V. Comparing SDC methods for microdata on the basis of information loss and disclosure risk. In Pre-proceedings of ETK-NTTS'2001, 2001, pp. 807–826.
2. Domingo-Ferrer J. and Torra V. A quantitative comparison of disclosure control methods for microdata. In P. Doyle, J.I. Lane, J.J.M. Theeuwes, and L. Zayatz (eds.). Confidentiality, Disclosure and Data Access: Theory and Practical Applications for Statistical Agencies. North-Holland, Amsterdam, 2001, pp. 111–134.
3. Yancey W.E., Winkler W.E., and Creecy R.H. Disclosure risk assessment in perturbative microdata protection. In J. Domingo-Ferrer (ed.). Inference Control in Statistical Databases. LNCS, Vol. 2316. Springer, 2002, pp. 135–152.

Search Engine Metrics

BEN CARTERETTE

University of Massachusetts Amherst, Amherst,
MA, USA

Synonyms

Evaluation measures; Performance measures

Definition

Search engine metrics measure the ability of an information retrieval system (such as a web search engine) to retrieve and rank relevant material in response to a user's query. In contrast to database retrieval, relevance in information retrieval depends on the natural language semantics of the query and document, and search engines can and do retrieve results that are not relevant. The two fundamental metrics are *recall*, measuring the ability of a search engine to find the relevant

material in the index, and *precision*, measuring its ability to place that relevant material high in the ranking. Precision and recall have been extended and adapted to many different types of evaluation and task, but remain the core of performance measurement.

Historical Background

Performance measurement of information retrieval systems began with Cleverdon and Mills in the early 1960s with the Cranfield tests of language indexing devices [4,3]. Prior to that, retrieval systems had been measured primarily by their efficiency; as with databases, it was implicitly assumed that any document matching the query was relevant. Cleverdon and Mills recognized that information retrieval is not like database retrieval. Queries can be under- or over-specified, polysemy can confound the relationship between query and document, the wrong word can be chosen for a concept with many names, and so on. Results that are not relevant to the user's request will be returned and results that are relevant will not be returned; there is a need to measure how often this can be expected to happen in general.

Cleverdon and Mills identified two primary dimensions on which to evaluate performance: the proportion of relevant material retrieved (the *recall ratio*) and the proportion of retrieved material that is relevant (originally the *relevance ratio*, later *precision*). Part of the goal of the Cranfield tests was to measure how different indexing strategies affected recall and precision. To this end, Cleverdon and Mills assembled a collection of 1,100 papers in high speed aerodynamics and asked the authors to list the research questions that inspired the paper. Each of the cited references was then judged for relevance to each of the questions. The resulting set of data – a collection of documents, a set of questions or queries, and judgments of the relevance of each document to each query – is called a *test collection*, and the use of test collections for information retrieval evaluation is now referred to as the Cranfield methodology.

Through his extensive evaluations of the SMART retrieval system in the 1960's and 1970's using the Cranfield collection and methodology, Gerald Salton cemented precision and recall as the primary evaluation metrics for search engine performance [8]. He additionally offered extensions and refinements to these basic measures: normalized precision and recall, precision-recall curves to demonstrate the tradeoff between the two,

interpolated precision at standard levels of recall, average precision over different levels of recall or different queries.

The early 1990s saw the formation of the Text Retrieval Conference (TREC) and the first evaluations over hundreds of thousands of full-text documents rather than the tens of thousands of abstracts that had previously been the standard in research [12]. The TREC collections are large and heterogeneous, and thus are a prime proving ground for any automatic retrieval technique. However, with an order-of-magnitudes increase in the number of documents, it became impossible to know every relevant document, and thus to know recall with certainty. TREC also motivated the birth of a field of research on "meta-evaluation," the evaluation of performance measures themselves, the evaluation of test collections, and the estimation of retrieval measures.

TREC also led the way in defining and providing models for the evaluation of new retrieval tasks. Some of the tasks studied and evaluated at TREC over the years include routing, multimedia retrieval, cross-language retrieval, and passage retrieval. Closely related to TREC are conferences on machine translation, summarization, and document understanding. Precision- and recall-based metrics such as the BLEU score [5] have become standard in these fields as well.

The 1990's also saw the explosive growth of the web, which over the past 15 years has grown into a collection of billions of documents, and within which search is a multi-million dollar industry. Accurate measures of performance are more important than ever, as millions of dollars are at stake when decisions are made based on those measures.

Foundations

Automatic text retrieval systems such as web search engines return results in the form of a ranked list, with the documents most likely to be relevant to the user's request at the top. Bad results can be returned if a query is over- or under-specified, a word with multiple meanings included in the query, or a word chosen to represent a concept that is in the index but not under that word, the ranked list will be "polluted" with nonrelevant results. To understand the extent to which this happens and how to fix it, it is necessary to evaluate the ability of the system to retrieve and rank relevant material independent of other factors affecting the utility of the search engine such as interface design or response time.

The two primary dimensions on which to evaluate a ranked list are its ability to find the indexed relevant material (recall) and its ability to rank that relevant material highly (precision). Formally, precision and recall are defined for a given rank cut-off in terms of binary relevance – each document is either relevant or not. Considering everything above the cut-off to be “retrieved” and everything below it to be “not retrieved” and comparing to the relevance of each document produces a 2×2 contingency table, as shown in Fig. 1.

Precision at rank n is defined as the proportion of relevant documents in the top n retrieved:

$$\text{precision}@n = \frac{\text{number of documents relevant \& retrieved in the top } n}{\text{number retrieved}}$$

Recall at rank n is the proportion of all relevant documents in the index retrieved in the top n :

$$\text{recall}@n = \frac{\text{number of documents relevant \& retrieved in the top } n}{\text{total number relevant}}$$

this means that the engine was able to find every relevant document without ever confusing a nonrelevant document for relevant.

As Fig. 2 shows, precision-recall curves appear jagged, as each new relevant document increases both precision and recall. The curve can be smoothed into a non-increasing curve by *interpolating* precision: k

equally-spaced points of recall are chosen, and the interpolated precision at the i th point is defined as the highest precision at any point of recall greater than or equal to that point [7]. The interpolated curve demonstrates the trade-off between recall and precision: retrieving more documents increases recall, but it also brings more nonrelevant material into the ranking, decreasing precision. Figure 2 shows an example of an 11-point (recall = 0,0.1,...,1) interpolated curve.

Besides precision and recall (which readers may also know as “positive predictive value” and “sensitivity” respectively) there are many other statistics that can be calculated on a 2×2 contingency table as in Fig. 1: specificity, χ^2 , mutual information, and accuracy, among others; besides the precision-recall curve, there are other curves that can be plotted over varying cut-off values, the ROC curve being the most famous. The utility of these to information retrieval is limited: they depend on counts of “true negatives,” i.e., nonrelevant documents that were not retrieved. When retrieving documents over a large heterogeneous collection such as the web, nonretrieved nonrelevant documents make up the vast majority of indexed pages – to a close approximation, 100% of the index. Thus the difference in one of these statistics for any two rankings is negligible, and certainly not distinguishable from chance.

A number of statistics that summarize the precision-recall curve have been invented over the years. The most common is *average precision*, the area under the precision-recall curve (originally the interpolated curve, now more commonly the non-interpolated curve).

The diagram illustrates a ranking of 8 documents (R for Relevant, N for Nonrelevant) and four corresponding 2x2 contingency tables for different rank cut-offs:

Rank	Document	Rank 1	Rank 3	Rank 5	Rank 7
1	R	Retrieved			
2	N	Not retrieved			
3	R	Retrieved	Retrieved		
4	N	Not retrieved	Not retrieved		
5	R	Retrieved	Retrieved	Retrieved	
6	N	Not retrieved	Not retrieved	Not retrieved	
7	R	Retrieved	Retrieved	Retrieved	Retrieved
8	N	Not retrieved	Not retrieved	Not retrieved	Not retrieved

Below the tables, the following 2x2 contingency tables are shown:

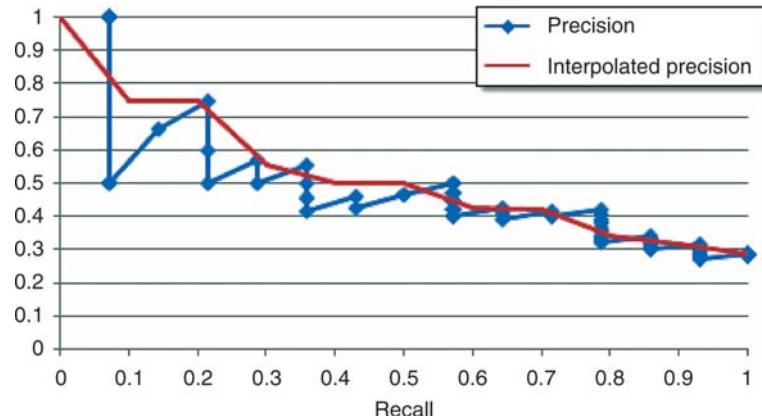
	Relevant	Nonrelevant
Retrieved	1	0
Not retrieved	3	4

	Relevant	Nonrelevant
Retrieved	2	1
Not retrieved	2	3

	Relevant	Nonrelevant
Retrieved	3	2
Not retrieved	1	2

	Relevant	Nonrelevant
Retrieved	4	3
Not retrieved	0	1

Search Engine Metrics. Figure 1. A ranking of eight documents, four of which have been judged relevant (R) and four nonrelevant (N). Cut-offs at ranks one, three, five, and seven produce the 2×2 tables shown.



Search Engine Metrics. Figure 2. An example precision-recall curve, along with its 11-point interpolated curve.

Another is *R-precision*, the point at which recall and precision are equal. These are both measures that reward systems for having both high recall and high precision, but in a nonlinear fashion. The F-measure is more linear: it is the weighted harmonic mean of precision and recall (weights are chosen depending on the relative importance of precision vs. recall); max-F is the highest of all such values.

Additionally, there are a number of other metrics in the literature that are based on the fundamental ideas behind precision and recall. One that has found widespread use in web measurement is *discounted cumulative gain* (DCG) [5]. Since it can handle graded relevance judgments, it is more flexible than precision as traditionally defined. *Normalized DCG* (NDCG) incorporates a recall component into DCG by dividing it by the best possible DCG for the query.

Relevance Judgments

As described above, calculating metrics requires judgments of the relevance of each document to each query. Precision requires a judgment on every retrieved document to the query. Recall requires that every document that is in the index and relevant to the query has been identified; thus until every document in the collection has been judged, the possibility remains that recall is being overestimated.

In Cleverdon and Mills' original experiments, judgments were made on how relevant a cited reference was to the research questions that inspired the citing paper [4]. They were made by the authors of the citing papers, the ones who came up with the research questions to begin with. To fill out the set needed for precise

recall computation, additional judgments were made by students working for Cleverdon.

With the shift to much larger, much more heterogeneous document collections that was inaugurated by the National Institute for Standards and Technology (NIST) at the TREC conferences, it became impossible to judge every document to every query. Instead, the *pooling* method [10] was adopted: a set of queries is sent to participating sites without relevance judgments; sites run the queries through their retrieval systems and return the resulting ranked lists to NIST. The top N documents retrieved by each system are pooled, and the entire pool judged for relevance. Although this results in a small fraction of the total collection being judged, it is a biased sample that ensures that most of the documents that are likely to be retrieved by any system will be judged. Zobel has shown that although relevant documents are missed using this method (and thus recall overestimated), it is more than satisfactory for evaluation when the goal is comparing two or more different systems [13]. Additional work has shown that reliable comparisons can be made with very few judgments; even when judgments needed to calculate precision are missing, system comparisons can often be made with high confidence, and even when confidence is not high, the degree of confidence can be reliably estimated [2,1].

The judgments acquired for TREC are typically binary – relevant or not – or trinary – highly relevant, relevant, or not relevant – but binarized for evaluation. This is appropriate for the tasks studied at TREC, which tend to emphasize recall. For many types of web searches, recall is significantly less important than

precision. For example, the query “microsoft” may return Microsoft’s corporate web page, pages about Microsoft software, pages about court cases Microsoft is involved in, and pages about Microsoft’s stock activity. All of these are relevant to some user’s need, but it is unlikely that all of them are relevant to the same need. Since the query is so broad, the best ranking would probably put Microsoft’s home page at rank 1. But if relevance is binary, any of those pages would be considered equally relevant, and a page about a small drop in Microsoft stock on a certain date could appear at rank 1 without affecting precision or recall, even though the user’s utility is clearly negatively affected.

To resolve this, web judgments are often made on a graded scale. Examples of graded scales include the “highly relevant,” “relevant,” and “nonrelevant” sometimes used at TREC; “highly relevant,” “relevant,” “maybe relevant,” “nonrelevant” to allow for some uncertainty on the part of the judge, or the five-point scale originally used by Cleverdon and Mills. There is a trade-off between finer performance distinctions and judgment quality, however: as more categories are added, it becomes harder to define what exactly distinguishes one category from another, and as a result the judgments become less reliable. Even with the binary judgments and highly-specified information needs used at TREC, there is a fair amount of disagreement about what is relevant [11]; when moving to finer scales and trying to infer user’s needs on the basis of a 1–3 word query, disagreement may skyrocket.

Hypothesis Testing and Relative Performance

Measures like average precision and NDCG defy easy interpretation. What does it mean for a system to have an NDCG of 0.69? Thus the goal of performance measurement is often to compare the performance of two engines, one of which may be a minor modification of the other. But a small difference in performance can occur simply by chance. A decision based on such a difference should take into account the probability that it is “real,” i.e., whether it is unlikely to have occurred only due to random factors.

Estimating this probability involves taking a random sample of queries likely to be input to the system. For the web, the sample can be obtained from search logs. The measure of interest is computed for each query, and some test statistic computed over the set. The ideal test statistic should have high power to detect the “real” differences when they exist.

There has been some debate over which test statistic (and therefore which hypothesis test) is applicable to information retrieval. If the same sample of queries can be treated as a random sample to either engine and both engines index the same documents, paired tests provide more powerful analysis [7]. The sign test makes no assumptions about the distribution of metrics like NDCG over queries, but is not very powerful. The Wilcoxon sign rank test, which has been popular, also makes no distributional assumptions, but as a test for difference in median has limited power to detect differences in mean performance. The t-test is a powerful test for detecting differences in means. Although it requires some distributional assumptions that may not hold in practice, it is robust to violations of those assumptions, and therefore is probably the best test to use when at least 25 queries can be sampled [13].

Key Applications

Measuring Search Engine Performance

Precision, recall, and DCG measure how well the engine ranks documents independent of other factors that can influence users’ opinions, such as interface, extra tools, and so on. Each metric measures a different aspect of performance with varying degrees of fineness.

Comparing Search Engines

Metrics allow the comparison of two different search engines or two variations on a baseline ranking algorithm. The statistical significance of differences can be evaluated and used to make decisions about development and deployment.

Optimizing Search Engine Performance

Search engine algorithms can be optimized to maximize performance on one or more of these metrics.

Future Directions

There are many open problems in search performance measurement: how to evaluate personalized search (in which results are tailored to the user), how to evaluate novelty (ensuring that the same information is not duplicated in results), how to use context in evaluation, and so on.

A challenge of web evaluation is the ever-changing nature of the query stream and the indexed documents [9]. The distribution of queries changes frequently, and

there is always a long tail of queries that only appear in the logs once. As a result, queries need to be resampled and reevaluated constantly. Web pages disappear or fall out-of-date frequently, and judgments should be kept accordingly up-to-date. Finally, changes in the search engine's interface or its underlying algorithms can affect the way users interact with it, making comparisons between engines separated by long time periods difficult if not impossible.

Finally, there is still more work to be done on understanding how missing relevance judgments affect conclusions that can be drawn from evaluations.

Data Sets

The TREC test collections described in this article are available from NIST at <http://trec.nist.gov/>.

Cross-references

- ▶ Average Precision
- ▶ Discounted Cumulated Gain
- ▶ F-Measure
- ▶ Information Retrieval
- ▶ MRR
- ▶ Relevance
- ▶ R-Precision
- ▶ Web Page Quality Metrics
- ▶ Web Search Relevance Ranking

Recommended Reading

1. Aslam J.A., Pavlu V., and Yilmaz E. A statistical method for system evaluation using incomplete judgments. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 541–548.
2. Carterette B., Allan J., and Sitaraman R.K. Minimal test collections for retrieval evaluation. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 268–275.
3. Cleverdon C.W. The cranfield tests on index language devices. In Readings in Information Retrieval. K.S. Jones and P. Willett (eds.). Morgan Kaufmann, 1967, pp. 47–59.
4. Cleverdon C.W. and Mills J. The testing of index language devices. In Readings in Information Retrieval. K.S. Jones and P. Willett (eds.). Morgan Kaufmann, 1963, pp. 98–110.
5. Kekalainen J. and Jarvelin K. Using graded relevance assessments in ir evaluation. JASIST, 53:1120–1129, 2002.
6. Papineni K., Roukos S., Ward T., and Zhu W.J. BLEU: a method for automatic evaluation of machine translation. In Proc. 40th Annual Meeting of the Assoc. for Computational Linguistics, 2002, pp. 311–318.
7. van Rijsbergen C.J. Information Retrieval. Butterworths, London, UK, 1979.

8. Salton G. and Lesk M.E. Computer evaluation of indexing and text processing. In Readings in Information Retrieval. K.S. Jones and P. Willett (eds.). Morgan Kaufmann, 1967, pp. 60–84.
9. Soboroff I. Dynamic test collections: measuring search effectiveness on the live web. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 276–283.
10. Sparck J.K. and van Rijsbergen C.J. Information retrieval test collections. J. Doc., 32(1):59–75, 1976.
11. Voorhees E. Variations in relevance judgments and the measurement of retrieval effectiveness. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 315–323.
12. Voorhees E.M. Harman D.K. (eds.). TREC: Experiment and Evaluation in Information Retrieval. MIT, Cambridge, MA, USA, 2005.
13. Zobel J. How reliable are the results of large-scale information retrieval experiments? In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 307–314.

Search Engine Query Result Caching

- ▶ Web Search Result Caching and Prefetching

Search Ranking

- ▶ Web Search Relevance Ranking

Searching Compressed XML

- ▶ Managing Compressed Structured Text

Searching Digital Libraries

PANAGIOTIS G. IPEIROTIS

New York University, New York, NY, USA

Synonyms

Federated search

Definition

Searching digital libraries refers to searching and retrieving information from remote databases of digitized or digital objects. These databases may hold

either the metadata for an object of interest (e.g., author and title), or a complete object such as a book or a video.

Historical Background

The initial efforts to standardize and facilitate searching of digital libraries date back to the 1970s, when the development of the Z39.50 protocol started. The Z39.50 protocol is an ANSI standard and defines how to search and retrieve items from a remote database catalog. The Z39.50 protocol was widely deployed within library environments, allowing users to perform searches to remote libraries.

With the advent of the Web, libraries started digitizing and making contents available on the Web, and the Z39.50 protocol started losing its importance. Many libraries made their content “searchable” through standard Web forms, allowing users to search and retrieve content using simply a Web browser. However, due to the lack of a link structure, the contents of the libraries remained “hidden” from the modern search engine crawlers, forming part of the “Hidden-Web” (also known as Deep Web, or Invisible Web). Searching across multiple Hidden Web databases, despite the tremendous progress since 2000, is still an open research problem.

However, achieving interoperability across all Web databases is inherently harder than achieving interoperability across library databases, which are relatively more homogeneous. Therefore, a set of efforts focused on introducing protocols to facilitate integrating and searching digital libraries. The Open Archives Initiative focused on defining a protocol for exporting metadata about the objects in the collections hosted by each library. The SRU protocol aims to modernize the Z39.50 by making it similar to modern Web services. Such efforts allow programmers to leverage their existing skills and develop easier tools for the library market.

Foundations

Digital libraries host a variety of digital objects, including, but not limited to, textual documents, images, sounds, videos, or even multimodal objects that combine the above. The concept of searching digital libraries may refer either to the action of searching a *single* digital library or to the action of searching across *multiple* digital libraries.

Searching a *single* digital library typically refers to the action of searching and browsing the contents of the underlying relational, textual, or multimedia database.

Searching across *multiple* digital libraries is a concept that evolved significantly over the years. The development of these efforts is broadly divided in three periods:

- *The pre-Web period (late 1970s–mid 1990s)*: Development of the Z39.50 standard.
- *The early-Web period (mid 1990s–early 2000s)*: Emergence of the Web, and increased accessibility of libraries over the Web.
- *The Web-services period (early 2000s–now)*: Definition of protocols for Web services, and development of library-focused search and discovery protocols.

The Pre-Web Period

The first attempts to define a standardized, common protocol for searching library databases date back to the 1970s. Then, the “Linked Systems Project” examined how to provide support for standardized access method to a small set of homogeneous, bibliographic databases. This effort led to the formation of a NISO committee in 1979, which after years of efforts defined the “American National Standard Z39.50, Information Retrieval Service Definition and Protocol Specifications for Library Applications” in 1987. The protocol was later revised in 1992, in 1995, and in 2003. (See [11] for a detailed history and timeline of the development of Z39.50.)

The Z39.50 protocol was designed as a client-server protocol, defining how the client can search and retrieve information from a remote database. The protocol supports a significant number of actions, including searching across individual fields, such as author, abstract, title, and so on. Unfortunately, the protocol did not mandate the implementation of several aspects of the specifications, allowing the developers to choose the aspects of the protocol to implement. This led to unexpected behavior of some systems, as the same query, executed over the same underlying content, could return very different results, depending on the implementation. Furthermore, the extremely heavy specification made it difficult for vendors to develop systems that were fully compatible with each other.

The Early-Web Period

The emergence of the Web changed significantly the way that digital libraries make their content available. Many libraries, perhaps encouraged by the *Digital Libraries Initiative* in 1994, started digitizing and making their content available over the Web. This meant that user could simply visit the Web site of a library and then, using simply Web forms, could query and browse the holdings of the library.

A significant fraction of these new digital libraries are only accessible via a search interface and the ability to browse through a static hyperlink structure is often missing. This means that the contents of these libraries are “hidden” from search engines, since traditional crawlers, which discover new pages by following links, cannot discover the contents of the library. Such libraries are part of the *hidden-Web* [2]. On the other hand, libraries that provide a link structure for accessing their holdings, are part of the *surface Web*, which is accessible by using general search engines, such as Google.

For libraries with content available as part of the *surface Web*, the common model for searching is through vertical search engines. The vertical search engines create topically-focused indexes of the material available on the Web by using *focused crawlers* [4] to identify and index the pages about a given topic. Under this model, the distributed digital libraries become searchable through a centralized search interface that indexes the remotely stored content. When a user issues a query, the vertical search engine identifies the most relevant pages in the index and returns to the user the URLs of the pages, which are stored remotely.

For libraries with *hidden Web* content, the typical way of searching their contents is through *metasearchers*. A complete metasearcher has to perform the following tasks:

- Discover the available digital libraries. This involves crawling the Web to identify pages with Web forms that are search interfaces for underlying databases [5].
- Understand the capabilities of the available query interface [1,13,16].
- Characterize the contents of the underlying database, typically by extracting a small sample of the stored contents through query-based sampling. The characterization may involve classifying the database into a topic hierarchy [6], extracting a

statistical summary of the content [3,8], or it may involve keeping the actual sample as a surrogate for the contents of the database [7,15].

- Use the database characterization to select the most promising databases for evaluating a given query [9,15].
- Evaluate the queries in the selected databases, retrieve, and merge the results from multiple databases into a single list [14].

An alternative approach to the distributed search technique adopted by metasearchers is to try to download *all* contents of a hidden Web database [12]. Once all the contents of the remote digital libraries are retrieved and stored locally, the problem of searching multiple digital libraries is reduced to the problem of searching a single, centralized database. One of the issues in this case is the need to periodically refresh the local copy with the most recent contents of the remote database [10].

The Web-Services Period

During the early-Web period, the problem of integrating and searching across digital libraries was similar to the problem of integrating Web databases at large. The vision of the *semantic Web* promised a solution for this problem, and the implementation of a *Web services* framework was a first step towards this direction.

Inherently, though, the library integration problem is much easier than the problems involved in the full implementation of the semantic Web. Therefore, a set of niche solutions were developed for the library integration problem, focusing on the one hand on library-specific needs, but building on top of the existing tools for general Web services that are being developed and rapidly improved.

One of the first attempts to make effortless the discovery of the contents of a library database was the development of the *Open Archives Initiative Protocol for Metadata Harvesting* (OAI-PMH). This protocol defines how a library can export metadata descriptions of its holdings. Then, *metadata harvesters* can easily collect the contents of the database and make these contents searchable through a centralized search interface. The OAI-PMH protocol is now widely adopted by many libraries and a set of OAI registries facilitate even further the discovery of libraries that support this protocol. Notably, major search engines, such as Google

and Yahoo! also support the protocol, as an alternative of the *sitemaps protocol*. This support allows libraries to be an integral part of the general Web and at the same time use a protocol developed and customized for their own needs.

Beyond OAI, there are also attempts to modernize the Z39.50 protocol and make it part of the larger family of Web protocols. First, the *Bath profile* specifies the exact query syntax that Z39.50 clients should use, so that clients can interpret the results returned by Bath-compliant Z39.50 servers. A more significant development is the agreement for the *Search/Retrieval via URL (SRU) protocol*. SRU is a standard XML-focused search protocol for Internet search queries that uses *Contextual Query Language (CQL)* for representing queries. The SRU uses the REST protocol and introduces a standard method for querying library databases, by simply submitting URL-based queries. For example, consider the following URL-encoded query:

<http://z3950.loc.gov:7090/voyager?version=1.1&operation=searchRetrieve&query=dinosaur&maximumRecords=10>

This example is a search for the term “dinosaur,” requesting that at most ten records to be returned. The SRU protocol is easy to support and implement, and is familiar to programmers that also use such syntax to interact with other popular Web services.

Key Applications

Digital libraries are increasingly becoming part of everyday life. The book digitization projects undertaken by corporations (e.g., Google, Microsoft) and by many universities will generate enormous digital archives accessible over the Web. Similarly, the high-quality holdings of the existing libraries are becoming increasingly accessible over the Web, allowing users to reach easier authoritative sources of information.

Cross-references

- [Bioinformatics Data Management](#)
- [Digital Libraries](#)
- [Health Informatics Databases](#)
- [Metadata Management](#)
- [Multimedia Databases](#)
- [Multimedia IR](#)
- [Querying over Data Integration Systems](#)
- [Scientific Databases](#)
- [Semantic Web and Ontology](#)
- [Semi-structured Text Retrieval](#)

- [Structured and Semi-structured Document Databases](#)
- [Text Retrieval](#)
- [Web Search and Crawl](#)
- [Web Services and Service Oriented Architecture](#)

Recommended Reading

1. Bergholz A. and Chidlovskii B. Using query probing to identify query language features on the web. In Distributed Multimedia Information Retrieval, In Proc. SIGIR 2003 Workshop on Distributed Information Retrieval, 2004, pp. 21–30.
2. Bergman M.K. The deep Web: surfacing hidden value. *J. Electron. Pub.*, 7(1), August 2001.
3. Callan J.P. and Connell M. Query-based sampling of text databases. *ACM Trans. Inf. Syst.*, 19(2):97–30, 2001.
4. Chakrabarti S., van den Berg M., and Dom B. Focused crawling: a new approach to topic-specific web resource discovery. *Comput. Netw.*, 31(11–16):1623–1640, May 1999.
5. Cope J., Craswell N., and Hawking D. Automated discovery of search interfaces on the web. In Proc. 14th Australasian Database Conf., 2003, pp. 181–189.
6. Gravano L., Ipeirotis P.G., and Sahami M. QProber: a system for automatic classification of hidden-web databases. *ACM Trans. Inf. Syst.*, 21(1):1–41, January 2003.
7. Hawking D. and Thomas P. Server selection methods in hybrid portal search. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 75–82.
8. Ipeirotis P.G. and Gravano L. Distributed search over the hidden web: hierarchical database sampling and selection. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 394–405.
9. Ipeirotis P.G. and Gravano L. When one sample is not enough: improving text database selection using shrinkage. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 767–778.
10. Ipeirotis P.G., Ntoulas A., Cho J., and Gravano L. Modeling and managing content changes in text databases. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 606–617.
11. Lynch C.A. The Z39.50 information retrieval standard. *D-Lib Mag.*, 3(4), April 1997.
12. Ntoulas A., Zerfos P., and Cho J. Downloading textual hidden web content by keyword queries. In Proc. ACM/IEEE Joint Conf. on Digital Libraries, 2005.
13. Raghavan S. and García-Molina H. Crawling the hidden web. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 129–138.
14. Si L. and Callan J. A semisupervised learning method to merge search engine results. *ACM Trans. Inf. Syst.*, 21(4):457–491, 2003.
15. Si L. and Callan J. Modeling search engine effectiveness for federated search. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 83–90.
16. Zhang Z., He B., and Chang K.C.-C. Understanding web query interfaces: best-effort parsing with hidden syntax. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 107–118.

Second Normal Form (2NF)

MARCELO ARENAS

Pontifical Catholic University of Chile, Santiago, Chile

Synonyms

[2NF](#)

Definition

Let $R(A_1, \dots, A_n)$ be a relation schema and Σ a set of functional dependencies over $R(A_1, \dots, A_n)$. An attribute A_i ($i \in \{1, \dots, n\}$) is a *prime* attribute if A_i is an element of some key of $R(A_1, \dots, A_n)$. Then specification (R, Σ) is said to be in Second Normal Form (2NF) if for every nontrivial functional dependency $X \rightarrow A$ implied by Σ , it holds that A is a prime attribute or X is not a proper subset of any (candidate) key for R [1].

Key Points

In order to avoid update anomalies in database schemas containing functional dependencies, 2NF was introduced by Codd in [1]. This normal form is defined in terms of the notions of prime attribute and key as shown above. For example, given a relation schema $R(A, B, C)$ and a set of functional dependencies $\Sigma = \{A \rightarrow B\}$, it does not hold that $(R(A, B, C), \Sigma)$ is in 2NF since B is not a prime attribute and A is a proper subset of the key AC . On the other hand, $(S(A, B, C), \Gamma)$ is in 2NF if $\Gamma = \{A \rightarrow B, B \rightarrow C\}$, since A is a key (and thus it is not a proper subset of any candidate key) and B is not contained in any (candidate) key for S .

It should be noticed that relation schema $S(A, B, C)$ above is in 2NF if $\Gamma = \{A \rightarrow B, B \rightarrow C\}$, although this schema is not in 3NF. In fact, 3NF is strictly stronger than 2NF; every schema in 3NF is in 2NF, but there exist schemas (as the one shown above) that are in 2NF but not in 3NF.

Cross-references

- ▶ [Boyce-Codd Normal Form](#)
- ▶ [Fourth Normal Form](#)
- ▶ [Normal Forms and Normalization](#)
- ▶ [Third Normal Form](#)

Recommended Reading

1. Codd E.F. Further Normalization of the Data Base Relational Model. In Data base systems. Englewood Cliffs, N.J. Prentice-Hall, 1972, pp. 33–64.

Secondary Index

YANNIS MANOLOPOULOS¹, YANNIS THEODORIDIS², VASSILIS J. TSOTRAS³

¹Aristotle University of Thessaloniki, Thessaloniki, Greece

²University of Piraeus, Piraeus, Greece

³University of California-Reverside, Riverside, CA, USA

Synonyms

[Non-clustering index](#)

Definition

A tree-based index is called a *secondary index* if the order which it maintains on the search-key values is *not* the same as the order of the file which it indexes. For example, consider a relation R with some numeric attribute A taking values over an (ordered) domain D . Assume that relation R is *not* physically stored on the values of attribute A (i.e., relation R is either stored as a heap – an unordered file, or is ordered on another attribute). Furthermore, assume that a tree-based index (e.g., B + -tree) has been created on attribute A . Then this index is secondary.

Key Points

Tree-based indices are built on numeric attributes and maintain an order among the indexed search-key values. They are further categorized by whether their search-key ordering is the same with the file's physical order (if any). Note that a file may or may not be ordered. Ordered is a file whose records are stored in pages according to the order of the values of an attribute. Obviously, a file can have at most a single such order since it is physically stored once. For example, if the *Employee* relation is ordered according to the *name* attribute, the values in the other attributes will not be in order. A file stored without any order is called an unordered file or heap. An index built on any non-ordering attribute of a file is called *secondary* (or non-clustering) while an index built on the ordering attribute of a file is called *primary* (clustering).

Since the actual data record can be anywhere in the file, the secondary index needs an extra level of indirection, namely, a pointer to the actual position of a record with a given value in the relation file. In other words, a secondary index only clusters references to

records (in the form of <value, pointer> fields), but *not* the records themselves. This extra indirection from a leaf page of a secondary index to the actual position of a record in a file has important subsequences on optimization. Consider, for example, a secondary index (B + -tree) on the *ssn* attribute of the *Employee* relation (which assume is ordered by the *name* attribute). A query that asks for the salaries of employees with *ssn* in the range $[x, y]$ can facilitate the B + -tree on *ssn* to retrieve references to all records in the query range. Assume there are 1,000 such *ssn* values in the *Employee* file. Since the actual *Employee* records must be retrieved (so as to report their salaries), each such reference needs to be materialized by possibly a separate page I/O (since the actual records can be in different pages of the *Employee* file).

A relation can have several indices, on different search-keys; among them, at most one is primary (clustering) index while the rest are secondary ones.

Cross-references

- ▶ Access Methods
- ▶ B+-Tree
- ▶ Index Sequential Access Method (ISAM)
- ▶ Indexing

Recommended Reading

1. Elmasri R.A., and Shamkant N.B. Fundamentals of Database Systems (5th edn.). Addison-Wesley, Reading, MA, 2007.
2. Manolopoulos, Theodoridis Y. and Tsotras Y. Vassilis J. Advanced Database Indexing. Kluwer, Dordrecht, 1999.
3. Ramakrishnan and Raghu Gehrke. Johannes Database Management Systems (3rd edn.). McGraw-Hill, NY, 2003.

Secret-Key Encryption

- ▶ Symmetric Encryption

Secure Data Outsourcing

BARBARA CARMINATI
University of Insubria, Varese, Italy

Synonyms

Secure third-party data management

Definition

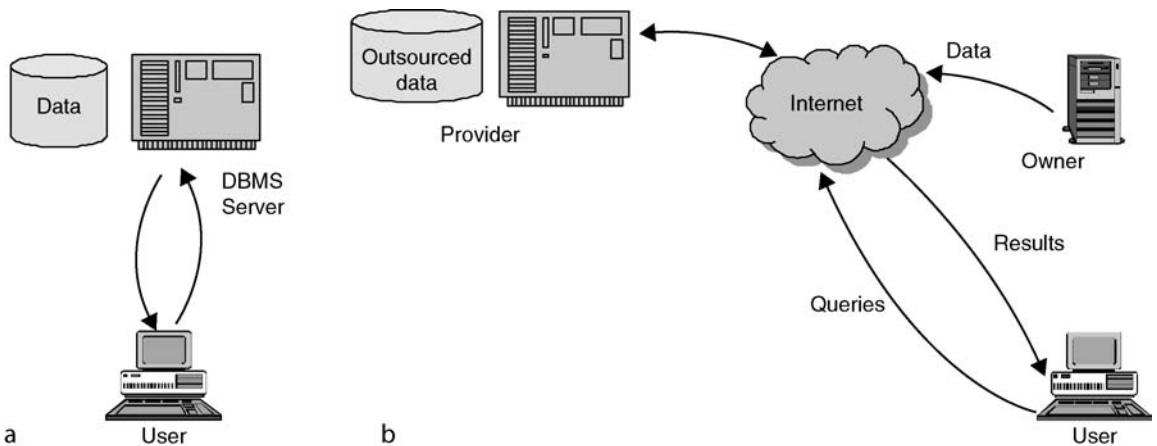
Data outsourcing is a new, emerging data management paradigm in which the owner of data is no longer totally responsible for its management. Rather, a portion of data is outsourced to external providers who offer data management functionalities. Secure data outsourcing is a discipline that investigates security issues associated with data outsourcing.

Historical Background

At present, service outsourcing is a paradigm widely used by many companies and organizations to achieve better service by delegating some of their business functions to external specialized service providers. A natural evolution of this paradigm is the recent emergence of *data outsourcing*. With this strategy, a company is no longer completely responsible for its own data management. Rather, it outsources some of its data functionalities to one or more external data management service providers (such as efficient query processing or large storage capability). Data outsourcing clearly leads to a range of security issues, because data owners have the potential to lose control over the data that is outsourced. Thus, the challenge is to ensure the highest level of security when data are managed by external service providers. With this aim, several research groups have started to investigate and propose mechanisms to achieve *secure data outsourcing*.

Foundations

In general, the enforcement of data outsourcing requires examination of new, challenging issues. With traditional, well-known client server architecture (see Fig. 1a), data owners manage the DBMS and directly answer user queries. Data outsourcing relies on third-party architecture (see Fig. 1b), in which data owners outsource their data (or portions of it) to one or more service providers. In real world environments, it cannot be assumed that third parties always operate according to the data owner's security policies. By contrast, to achieve secure data outsourcing, one needs to define techniques that satisfy the main security properties even in the presence of an *untrusted* third party – that is, a provider that could maliciously modify or delete the data it manages by, for instance, inserting fake records or sending data to unauthorized users. Several researchers have focused on this problem and have developed different proposals. However, before illustrating the techniques proposed to date, it is



Secure Data Outsourcing. Figure 1. (a) Two-party; (b) third-party architecture.

necessary first to identify the main security requirements in secure data outsourcing.

Security Requirements in Secure Data Outsourcing

In third-party architectures, potentially untrusted providers manage data. For this reason, secure data outsourcing must examine novel security issues as well as reexamine the traditional ones. The following presents the main security issues studied so far in secure data outsourcing.

Privacy: If a third-party architecture is adopted, a user could be concerned about his/her privacy for any query processing that is performed by a third-party provider. This is due to the fact that by simply tracking a user's queries, an untrusted provider could infer sensitive information about the user (for instance, the user's preferences). For this reason, the privacy of the submitted queries needs to be protected. To ensure access privacy, a provider should not be able to know the details of the query, but should be able to process it.

Authenticity and Integrity: Ensuring authenticity and integrity in third-party architecture enables a user, upon receiving some data from a provider, to verify that the data received has been in fact generated by the data owner and not modified by the provider. (*Integrity* also has an additional meaning, that is, ensuring that unauthorized users have not modified the data; however, since data outsourcing is mainly conceived for read-only data access, this definition of *integrity* is not considered in this entry.) In traditional architectures, both authenticity and integrity are ensured by means of digital signatures. When a user submits a query, the data owner re-evaluates it and

digitally signs the query result. Then, the query result together with its digital signature is sent to the user, thus enabling the user to verify the query's authenticity and integrity. However, in third-party architectures, traditional signature techniques cannot be used. A provider may return to the user only selected portions of the signed data in answer to the query evaluation. Thus, a user that is provided with only these portions is not able to validate the owner's digital signature, which has been generated on the whole data. To cope with these requirements, alternative ways must be found to sign outsourced data digitally so that a user is able to validate the digital signature even if he/she has only received selected portions of the signed data.

Completeness: Third-party architectures introduce a further novel security requirement, called completeness. If satisfied, this property ensures a user that the answer received by third-party service providers to a query genuinely contains all of the data answering to the submitted query. (In the literature, some works refer to this property as *query correctness*, by also implying authenticity and integrity requirements.)

Confidentiality: Data confidentiality means ensuring that data are disclosed only to authorized users. However, it is obvious that when data are outsourced, confidentiality requirements are not limited to users, but extended also to providers. Thus, confidentiality in data outsourcing acquires a twofold meaning. The first deals with protecting the owner's data from access by a malicious or untrusted provider, and is referred to as *confidentiality wrt the provider*. A further confidentiality requirement, hereafter called *confidentiality wrt users*, refers to the protection of data from

unauthorized user access on the basis of the access control policies stated by the data owners. In traditional client-server architectures, this requirement is enforced by access control mechanisms, called *reference monitors*, which mediate each user request by authorizing only those in accordance with the owner's access control policies. This type of solution can hardly be applied in data outsourcing, since it implies the delegation of the reference monitor tasks to a potentially untrusted publisher. For this reason, alternative solutions should be devised for access control enforcement when data services are outsourced.

Techniques for Secure Data Outsourcing

Many research groups have investigated security property enforcement in data outsourcing, resulting in several proposals for different security requirements. The following presents the main results proposed so far, grouped according to the security properties addressed by each.

Access Privacy: Private Information Retrieval protocols (PIR, for short), first introduced in [5], are one of the most relevant results of investigations into the problem query privacy protection. The underlying idea of PIR protocols is that several replications of the same database are available in different servers (i.e., providers). To preserve access privacy, the user submits different queries to each different server, defined so that by combining the servers' answers, the user is able to obtain the information desired, but by analyzing the submitted query, each server is unable to infer the actual interest of the user.

Authenticity and Integrity: These are the first properties that have been investigated in secure data outsourcing. The aim is to devise alternative digital signature schemes for signing the data to be outsourced, enabling users to validate the signature even if users are only provided with selected portions of the signed data. Several schemes have been proposed so far, exploiting different strategies for achieving this result. In particular, two of the most widely used techniques are Merkle trees and aggregate signatures. The following presents both of these concepts by introducing some of the related proposals.

Merkle Trees. Merkle proposed a method to authenticate, with a unique signature, a set of messages $\{m_1, \dots, m_n\}$, by at the same time enabling an intended verifier to authenticate a single message without the disclosure of the other messages. The proposed

solution exploits a binary tree, where each leaf contains the hash values of a message in $\{m_1, \dots, m_n\}$, whereas internal nodes enclose the concatenation of the hash values corresponding to its left and right children (see Merkle tree entry for more details). The root node of the resulting binary hash tree can be considered the digest of all messages, and thus it can be digitally signed by using a standard signature technique. The main benefit of this method is that a user is able to validate the signature by having a subset of messages, provided that he/she receives a set of additional hash values. Indeed, by having hash values of the missing messages, a user is able to build up locally the binary hash tree and thus is able to validate the signature. Merkle trees have been used in several computer areas. However, the first work to exploit them in data outsourcing was the one by Devanbu et al. [7], which adapts these trees to relational data to prove the completeness, authenticity, and integrity of query answers. According to this approach, for each relation R , a different Merkle tree is generated in such a way that leaves contain hash values of tuples in R . Then, when a user submits a query on R , the provider replies to him/her with the tuples answering the submitted query together with the signature generated on the root node of the corresponding Merkle tree. Moreover, the provider also sends the user the hash values of the tuples of R not included in the result set. These additional hash values enable the user locally to generate the Merkle tree of R and to validate the signature. A similar approach has been proposed to authenticate query results in edge computing [14]. Here, besides signing only the root of the Merkle tree, all leaves as well as all of the internal nodes are also signed. This leads to a reduced number of hash values to be returned to users to enable them to verify owner signatures. Merkle trees have also been investigated for secure data outsourcing of XML documents [1,6]. In these approaches, Merkle trees are generated in a different way wrt those computed over relational data, i.e., a flat list of tuples. Here, the challenge is how to generate a Merkle tree that exploits the hierarchical organization of an XML document. In both these approaches, a hash value is univocally associated with the document root through a recursive bottom-up computation on its structure. The digest is computed by associating a hash value with each node n of the XML document, computed by also taking into account the hash values of its children and the attributes in addition to contents of

the n itself. Thus, the digest of the whole document is the hash value of the root of the document.

Aggregate signatures. Boneh et al. [2] introduced the notion of aggregate signatures, that is, a signature scheme able to aggregates n distinct signatures generated by n distinct data owners into a unique digital signature. The main advantage of this scheme is that the validation of this unique digital signature implies the validation of each component signature. The aggregate signature has been used by Mykletun et al. in [12] to ensure authenticity and integrity of outsourced relational data. According to this approach, given a relation R , the data owner generates a different signature for each tuple in R . These signatures, together with the corresponding tuples, are outsourced to the provider. Then, when a user submits a query on relation R , the provider evaluates the query on R and aggregates all of the signatures corresponding to the tuples in the result set into a unique signature. Therefore, as result of the submitted query, the third party returns to the user the resulting aggregate signature, as well as the tuples answering the query. The properties of aggregate signatures assure the user that if the aggregate signature generated by the third party is valid, then all of the signatures generated by the owner on the tuples in the result set are also valid, which proves tuples' authenticity and integrity.

Completeness: Researchers have investigated the completeness property in combination with authenticity and integrity requirements. As a consequence, the main solutions proposed for ensuring this property exploit the same techniques used for authenticity and integrity – that is, Merkle trees and aggregate signatures. First, let introduce how completeness can be ensured by exploiting Merkle trees. In particular, [7] considers the problem of completeness of answers to range queries. The proposed solution's underlying idea is that given a range query whose predicate is against attribute a , the Merkle tree is generated over tuples sorted according to a values. Then, when a user submits a range query containing a predicate against a , the provider returns two additional tuples to the user together with the result set answering the range query – that is, the tuples precedent and subsequent to the lower and upper bound of the result set. These two values are then used to verify the owner's signature. Since the signature is computed on a Merkle tree generated over sorted tuples, if the user validates the owner's signature, he/she is ensured that the received precedent and

subsequent tuples really precede and follow the lower and upper bounds of the result set and that no tuples are omitted from the result set, which proves the completeness of the answer. In contrast, the approach to ensure completeness that has been proposed by Mykletun et al. [12] modifies an aggregation signature scheme in such a way to include in the signature of a tuple t , the hash value of tuples preceding t according to all possible sorts defined on a the relation's attributes, obtaining a so-called signature chain. Thus, when a user submits a range query, the provider also inserts into the result set the boundary tuples, i.e., the tuples preceding the upper and lower bound of the result set, as well as their aggregated signatures. By means of these signatures, a user is therefore able to prove that the third party has not omitted any tuple.

Notice that to ensure completeness, both these solutions require the user to be sent two additional tuples wrt the result set answering the range query. This could lead to some confidentiality breaches, since these additional tuples could contain sensitive data. To overcome this problem, Pang et al. [13] proposed an approach exploiting signature chains that does not disclose more tuples than those in the result set.

Confidentiality: Several research groups have investigated confidentiality issues in data outsourcing. Most have focused on confidentiality wrt providers, whereas only the approach proposed in [3] also investigates confidentiality wrt users in the context of outsourcing XML documents. It is interesting to note, however, that all these solutions share a common underlying idea by which the data owners provide outsource service providers with an encrypted version of the data to manage, without providing them with the corresponding decryption keys. Consequently, the third party is unable to access and to misuse outsourced data. This obviously ensures confidentiality wrt providers. Also, to provide assurance of confidentiality wrt users, the authors of [3] proposed a selective encryption for XML documents in which different portions of the same document are encrypted with different encryption keys. More precisely, the owner encrypts all portions of the data to which the same policies apply with the same key. Then, the owner provides each user with all and only the keys corresponding to the portions of data that the user is allowed to access. This selective key distribution ensures that each user is able to access all and only the portions of data for which there is an access control policy authorizing the access. Obviously, applying these solutions

requires addressing an interesting problem – that is, how to enable providers to evaluate queries on encrypted data without accessing them. In recent years, this problem has been deeply investigated by several researchers, with the results of different proposals. The following sections introduce some of them by grouping them according to the underlying data model.

Querying encrypted relational data. The most relevant solution to querying encrypted relational data has been proposed by Hacigumus et al. [10,11], where binning techniques and privacy homomorphic encryption are exploited to execute SQL queries over encrypted relations. The first step is to introduce how binning techniques enable a service provider to evaluate selection queries. The underlying idea in [10,11] is that, given a relation R , the data owner partitions the domain of each attribute in R into distinguished intervals, to which the data owner assigns a different id. Then, for each tuple t in R , the data owner outsources to providers its encryption complemented with the ids associated with the intervals to which t 's attribute values belong. According to this approach, when a user intends to submit a query, he/she rewrites the query's conditions in terms of interval ids, thus enabling the provider with the ability to evaluate them without accessing the data. For instance, the condition “Salary =200K” is rewritten as “Salary = $id(200K)$,” where $id(200K)$ is the id of the partition containing the value 200K. Moreover, to enable third parties to evaluate aggregate functions over encrypted data, in [11] Hacigumus et al. exploited privacy homomorphisms (PH) to calculate some arithmetic operations directly on encrypted data. More precisely, PH functions are used to encrypt attributes of R , on which it is expected to do some aggregations. The data owner outsources the encrypted tuples of R and the corresponding partition ids together with attributes encrypted by PH functions. In [11], it is shown that by having the privacy homomorphisms of attributes to be aggregated, the third party is able to evaluate aggregate functions directly over them.

Querying encrypted textual data. In 2000, Song et al. proposed a first cryptographic scheme that supports searching words on encrypted textual data [16]. (In this context, the textual data consists of a set of encrypted words.) According to this scheme, the third party is provided with a ciphered version of words to be managed. These ciphered words are generated as follows: first, each word is symmetrically encrypted with a single secret key k ; then, each

resulting encrypted word is XORed with a different pseudorandom number. Since different occurrences of the same encrypted word are XORed with a different pseudorandom number, information about the word distribution cannot be inferred by analyzing the distribution of the encrypted words. Each user is provided with the secret key k , and the used pseudorandom numbers. (The scheme proposed in [16] is defined in such a way that users are able to compute pseudorandom numbers locally without any interaction with the data owner.) By having this information, therefore, when a user intends to ask the third party for a keyword W , the user first generates the encrypted word using the secret key k , and then computes the XOR of the result with the corresponding pseudorandom number. The user then submits the obtained ciphered word to the third party, which sequentially scans all ciphered words to search for the one matching the one submitted. Thus, this scheme allows the third party to search for a keyword W directly on the ciphered data without gaining any information on the clear text or on the required keyword W . In 2003, Eu-Jin Gon proposed an alternative solution for searching keywords in an encrypted document [8], based on indexes. According to this approach, a different index is associated with each document to be encrypted. These indexes, called security indexes, are based on Bloom filters and have the property to store hidden information about the keywords contained within the corresponding document. According to this scheme, the owner outsources the encrypted documents and the corresponding security indexes to service providers, which are then able to search for a keyword by simply accessing the indexes. A similar approach has been devised in [4], which proposes dictionary-based keyword indexes.

However, all of this work has the limitation that third parties are able to identify only documents matching with a given keyword, but are not able to support more expressive searches, such as Boolean combinations of keywords. A first step to overcome this limitation has been done by Golle et al. in [9], which proposes a public key scheme to support conjunctive keywords searches.

Key Applications

Data outsourcing offers several benefits. One of the most relevant is related to cost reduction. Indeed, the company pays only for services that it uses from providers, which are generally significantly less than the cost implied by deployment, installation, maintenance, and

upgrades of DBMSs. Moreover, the data management services offered by specialized providers are more competitive than the ones provided by the company itself. A further benefit of data outsourcing is its scalability, since a company can outsource its data to as many providers as it needs according to the amount of data and the number of managed users, avoiding that provider might become a bottleneck for the system. All these benefits make secure data outsourcing suitable for a wide range of applications in different data domains. For geographical data, for instance, the secure data outsourcing paradigm can be adopted to support geo-marketing services. A data owner can outsource some of its geographical data (for instance, maps at various levels of details) to a publisher that provides them to customers based upon different registration fees or different confidentiality requirements (for instance, maps of some regions that cannot be distributed to everyone because they show sensible objectives).

Future Directions

Given the attention that the data outsourcing paradigm is receiving, it is expected that secure data outsourcing will be intensely investigated in the future. Besides proposing more efficient strategies for the security requirements considered so far, it is necessary to consider further challenging security issues, like those related to user privacy and ownership protection. Moreover, more consideration must be given to complex data outsourcing scenarios to enable users to manipulate the outsourced data rather than just simply reading it.

Cross-references

- ▶ Access Control
- ▶ Data Encryption
- ▶ Digital Signatures
- ▶ Merkle Trees

Recommended Reading

1. Bertino E., Carminati B., Ferrari E., Thuraisingham B., and Gupta A. Selective and authentic third-party distribution of XML documents. *IEEE Trans. Knowl. Data Eng.*, 16(10):1263–1278, 2004.
2. Boneh D., Gentry C., Lynn B., and Shacham H. Aggregate and verifiably encrypted signatures from bilinear maps. In Proc. Advances in Cryptology, 2003.
3. Carminati B., Ferrari E., and Bertino E. Securing XML data in third-party distribution systems. In Proc. Int. Conf. on Information and Knowledge Management, 2005.

4. Chang Y. and Mitzenmacher M. Privacy preserving keyword searches on remote encrypted data, Cryptology ePrint Archive, Report, 2004.
5. Chor B., Goldreich O., Kushilevitz E., and Sudan M. Private information retrieval. In Proc. Symp. on Foundations of Computer Science, 1995.
6. Devanbu P., Gertz M., Kwong A., Martel C., Nuckolls G., and Stubblebine S.G. Flexible authentication of XML documents. In Proc. 8th ACM Conf. on Computer and Communications Security, 2001.
7. Devanbu P., Gertz M., Martel C., and Stubblebine S.G. Authentic third-party data publication. In Proc. 14th Annual IFIP WG 11.3 Working conference on Database Security, 2000.
8. Goh E. Secure Indexes, Cryptology ePrint Archive, Report 2003/216, 2003.
9. Golle P., Staddon J., and Waters B. Secure conjunctive keyword search over encrypted data. In Proc. the Applied Cryptography and Network Security Conf., 2004.
10. Hacigumus H., Iyer B., Li C., and Mehrotra S. Executing SQL over encrypted data in the database service provider model. In Proc. 9th Int. Conf. on Database Systems for Advanced Applications, 2002.
11. Hacigumus H., Iyer B., Li C., and Mehrotra S. Efficient execution of aggregation queries over encrypted relational databases. In Proc. 9th Int. Conf. on Database Systems for Advanced Applications, 2004.
12. Mykletun E., Narasimha M., and Tsudik G. Authentication and integrity in outsourced databases. In Proc. 11th Annual Symp. on Network and Distributed System Security, 2004.
13. Pang H., Jain A., Ramamirtham K., and Tan K. Verifying completeness of relational query results in data publishing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
14. Pang H. and Tan K. Authenticating query results in edge computing. In Proc. 20th Int. Conf. on Data Engineering, 2004.
15. Rivest R., Adleman L., and Dertouzos M. On data banks and privacy homomorphisms. In Foundations of Secure Computation, Richard J. Lipton, David P. Dobkin, Anita K. Jones (eds.). Academic press, 1978, pp 169–178.
16. Song D.X., Wagner D., and Perrig A. Practical techniques for searches on encrypted data. In Proc. IEEE Symp. on Security and Privacy, 2000.

Secure Database Design

- ▶ Secure Database Development

Secure Database Development

JAN JURJENS¹, EDUARDO B. FERNANDEZ²

¹The Open University, Buckinghamshire, UK

²Florida Atlantic University, Boca Raton, FL, USA

Synonyms

Secure DBMS development; Secure database design

Definition

This entry considers how to build secure database system software. In particular, it describes how to build a general-purpose database management system where security is an important design parameter. For the database community, the words secure database design may refer to the schema design to produce a database for a specific application with some level of security properties. There is a large amount of literature on this latter subject and a related entry in this encyclopedia (Database security). This entry concentrates mostly on how to build the software of a DBMS such that it exhibits security properties, which is called secure database development. Both approaches are contrasted so that the reader can decide which one of these problems applies to their specific case but more space is dedicated to the general secure database development problem.

Historical Background

While there is a large number of papers on security models including authorization and other security aspects of databases [2,4,5], there is little work on how to implement a secure Database Management System (DBMS). It is true that many proposals for secure multilevel databases include details of implementation but most of them are ad hoc architectures that cannot be generalized to databases using different models or even to other multilevel databases with different requirements. Of the books on database security, [7] had several chapters on how to build secure relational database systems, and later [4] included also multilevel models. Those books do a good job of indicating the architectural units of such systems and their general requirements. However, software development aspects are not discussed in detail. It appears that [10] is the only work discussing these aspects explicitly.

Foundations

There are two aspects to the problem of developing software for secure databases: building a general (application-independent) secure DBMS and building a database system which is part of a secure application. These two problems are first briefly defined and then discussed in more detail. Other approaches and possible system architectures are also considered.

In the first approach the DBMS is just a complex software application in itself and a general secure software methodology can be applied without or with little change. Object-oriented applications typically start from

a set of use cases, which define the user interactions with the system under development. In this particular case, use cases would define the typical functions of a DBMS, e.g. search, query, and update, and security would be included as part of its development life cycle. The DBMS would follow an appropriate model, e.g. Role-Based Access Control (RBAC), selectable in the design stage, which defines security constraints for the functions defined by the use cases. In some cases, it may be possible to support more than one security model. This would result in a secure DBMS where security would be a general nonfunctional requirement. The approach results in a general-purpose secure DBMS, where nothing is known about the specific applications that will be executed by its users. The DBMS itself is the application. The secure development methodologies of [6,13] and others are applicable here.

Another view is the one from a designer who needs to build a specific user application (or type of application) that includes a DBMS as part of its architecture, e.g. a financial system (most applications require a database but the degree of security needed may vary). This is discussed in [4,9,10,11]. In this case, the DBMS is rather ad hoc and tailored to the level of security desired for the specific type of application. For example, [11] separates the requirements into three types: functional, security, and database. Typically, these approaches emphasize how to define and enforce a set of application-specific rules that follow some security model and how to reflect them in the schema and other parts of the DBMS. Most of these studies emphasize the security of the database schema or some specific sections without much concern for the rest of the application. A methodology such as [6] or [13] can also be applied here, the DBMS being one of the architectural levels of a system that implements a specific application, although these methodologies have little to say about the contents of the specific rules that are needed in the schema (only their safe storage but not their consistency or security).

An interesting problem that applies to both approaches is the mapping from the conceptual security model (that may apply to a collection of DBMSs) to the authorization system of a specific database; for example, security constraints defined in a conceptual UML model defining authorizations in terms of classes must be mapped to an SQL-based authorization system which defines authorizations in terms of relations. Clearly, whatever is defined in the common conceptual model must be respected in the DBMS authorization

system, although this latter may add further constraints related to implementation aspects.

General Secure Database Systems

In this case, as indicated earlier, the DBMS is a complex application requiring a general high level of security. There are several methodologies for this purpose and two of them are described below. A methodology for secure software development should include appropriate tools and provide a unified and consistent approach through all the life cycle stages. Ideally, a methodology should use a Model-Driven Development approach, where transformations between development stages are based on corresponding metamodels. Since the resulting software is independent of the access control model adopted, it does not provide for special requirements of the model; for example, multilevel models typically require data labeling. This means that the resulting software would be less secure than an ad hoc design (unless the multilevel model was the target in the example). Because of the generality of the resultant DBMS it may be difficult to prove formally security properties. An early approach in this direction was based on adding security functions to a general-purpose DBMS, e.g. INGRES or System/R.

Secure Database Development using Patterns

A methodology to build secure systems is presented in [6]. A main idea in the proposed methodology is that security principles should be applied at every stage of the software lifecycle and that each stage can be tested for compliance with those principles. Another basic idea is the use of patterns at each stage. A pattern is an encapsulated solution to a recurrent problem and their use can improve the reusability and quality of software.

Domain analysis stage: A business model is defined. Legacy systems are identified and their security implications analyzed. Domain and regulatory constraints are identified and used as global policies. The suitability of the development team is assessed, possibly leading to added training. This phase may be performed only once for each new domain or team. The need for specialized database architectures should be determined at this point. The approach (general DBMS or application-oriented system) should also be defined at this stage.

Requirements stage: Use cases define the required interactions with the system. Each activity within a

use case is analyzed to see which threats are possible. Activity diagrams indicate created objects and are a good way to determine which data should be protected. Since many possible threats may be identified, risk analysis helps to prune them according to their impact and probability of occurrence. Any requirements for degree of security should be expressed as part of the use cases.

Analysis stage: Analysis patterns can be used to build the conceptual model in a more reliable and efficient way. The policies defined in the requirements can now be expressed as abstract security models, e.g. access matrix. The model selected must correspond to the type of application; for example, multilevel models have not been successful for medical applications. One can build a conceptual model where repeated applications of a security model pattern realize the rights determined from use cases. In fact, analysis patterns can be built with predefined authorizations according to the roles in their use cases. Patterns for authentication, logging, and secure channels are also specified at this level. Note that the model and the security patterns should define precisely the requirements of the problem, not its software solution. UML is a good semi-formal approach for defining policies, avoiding the need for ad-hoc policy languages. The addition of OCL (Object Constraint Language) can make the approach more formal.

Design stage: When one has defined the policies needed, one can select mechanisms to stop attacks that would violate them. A specific security model, e.g. RBAC, is now implemented in terms of software units. User interfaces should correspond to use cases and may be used to enforce the authorizations defined in the analysis stage. Secure interfaces enforce authorizations when users interact with the system. Components can be secured by using authorization rules for Java or .NET components. Distribution provides another dimension where security restrictions can be applied. Deployment diagrams can define secure configurations to be used by security administrators. A multilayer architecture is needed to enforce the security constraints defined at the application level. In each level, one can use patterns to represent appropriate security mechanisms. Security constraints must be mapped between levels.

The persistent aspects of the conceptual model are typically mapped into relational databases. The design of the database architecture is done according to the requirements from the use cases for the level of

security needed and the security model adopted in the analysis stage. Two basic choices for the enforcement mechanism include query modification as in INGRES and views as in System R. A tradeoff is using an existing DBMS as a Commercial Off-the-Shelf (COTS) component, although in this case security will depend on the security of that component.

Implementation stage: This stage requires reflecting in the code the security rules defined in the design stage. Because these rules are expressed as classes, associations, and constraints, they can be implemented as classes in object-oriented languages. In this stage one can also select specific security packages or COTS, e.g., a firewall product or a cryptographic package. Some of the patterns identified earlier in the cycle can be replaced by COTS (these can be tested to see if they include a similar pattern). Performance aspects become now important and may require iterations. As indicated, a whole DBMS could be such component.

An important aspect for the complete design is assurance. Experience shows that one can verify each pattern used but this does not in general verify their combination. One can however still argue that since one has used a careful and systematic methodology with verified and tested patterns, the design should provide a good level of security. The set of patterns can be shown to be able to stop or mitigate the identified threats.

Secure Database Development using UMLsec

A general methodology for developing security-critical software which in particular can be used to develop secure DBMSs has been proposed in [13]. It makes use of an extension of the Unified Modeling Language (UML) to include security-relevant information, which is called UMLsec. The approach is supported by extensive automated tool-support for performing a security analysis of the UMLsec models against the security requirements that are included [14] and has been used in a variety of industrial projects [3]. The UMLsec extension is given in form of a UML profile using the standard UML extension mechanisms. Stereotypes are used together with tags to formulate the security requirements and assumptions. Constraints give criteria that determine whether the requirements are met by the system design, by referring to a precise semantics of the used fragment of UML. The security-relevant information added using stereotypes includes security assumptions on the physical level of the system, security requirements related to the secure handling and communication of data, and

security policies that system parts are supposed to obey. The UMLsec tool-support can be used to check the constraints associated with UMLsec stereotypes mechanically, based on XMI output of the diagrams from the UML drawing tool in use. There is also a framework for implementing verification routines for the constraints associated with the UMLsec stereotypes. Thus advanced users of the UMLsec approach can use this framework to implement verification routines for the constraints of self-defined stereotypes. The semantics for the fragment of UML used for UMLsec is defined using so-called UML Machines, which is a kind of state machine which is equipped with UML-type communication mechanisms. On this basis, important security requirements such as secrecy, integrity, authenticity, and secure information flow are defined.

Applications Including Secure Databases

Since this approach is tailored to the application, one can add the required level of security using formal proofs when necessary. Specialized operating system and hardware are also possible and may be needed to reach the required level of security. High-security systems require faithful application of basic security principles; for example, multilevel databases apply complete mediation. Databases work through transactions and a concurrency control system serializes transactions to prevent inconsistencies. High-security multilevel databases also require that the concurrency control system preserves security. The methods described in the last section still apply here, except that additional requirements must be considered. Because of this, these approaches are discussed in less detail, describing only two recent papers that contain references to past work.

Designing Secure Databases using OCL

An approach to designing the content of a security-critical data base uses the Object Constraint Language (OCL) which is an optional part of the Unified Modeling Language (UML). More specifically, [8] presents the Object Security Constraint Language V.2. (OSCL2), which is based in OCL. This OCL extension can be used to incorporate security information and constraints in a Platform Independent Model (PIM) given as a UML class model. The information from the PIM is then translated into a Platform Specific Model (PSM) given as a multilevel relational model. This can then be implemented in a particular Database Management System (DBMS), such as Oracle9i Label Security. These

transformations can be done automatically or semi-automatically using OSCL2 compilers. Related to this, [9] presents a methodology that consists of four stages: requirements gathering; database analysis; multilevel relational logical design; and specific logical design. Here, the first three stages define activities to analyze and design a secure database. The last stage consists of activities that adapt the general secure data model to one of the most popular secure database management systems: Oracle9i Label Security. They later extended the approach to data warehouses, multi-dimensional databases, and on-line analytical processing applications.

In both cases, a particular multilevel database system, meaning a set of users organized in levels, compartments, and groups, is given access to specific items of a relational database, according to the characteristics of those items, which also include levels, compartments, and groups. A set of rules describes the allowed access of users to data items. The secure metamodel is stored in the labels of each row or user definition. As indicated, the extra requirements can be superimposed in a general secure software development methodology.

Other Approaches to Secure Software Development with Applicability to Databases

There are other approaches to developing security-critical software which can be applied to

developing secure databases and database management systems.

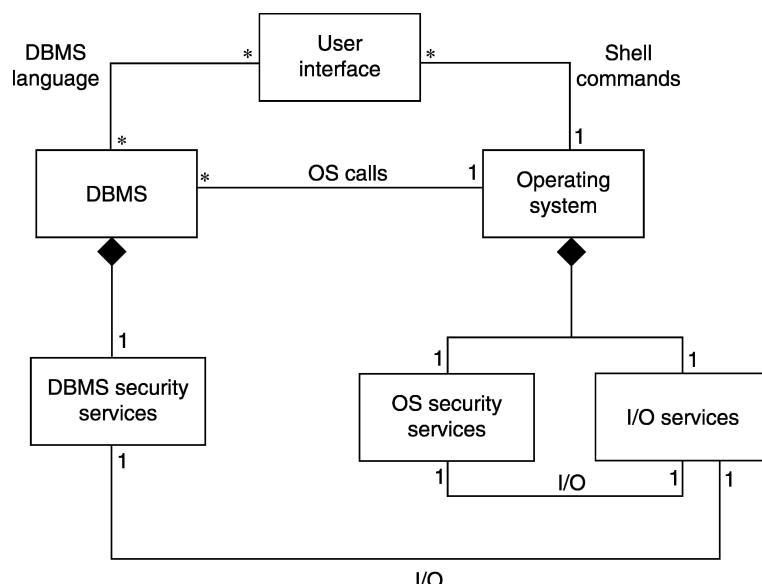
[12] presents an approach for the predicative specification of user rights in the context of an object oriented use case driven development process. It extends the specification of methods by a permission section describing the right of some actor to call the method of an object. The syntactic and semantic framework is first-order logic with a built-in notion of objects and classes provided with an algebraic semantics. The approach can be realized in OCL.

[1] presents an approach to building secure systems where designers specify system models along with their security requirements and use tools to automatically generate system architectures from the models, including complete, configured access control infrastructures. It includes a combination of UML-based modeling languages with a security modeling language for formalizing access control requirements.

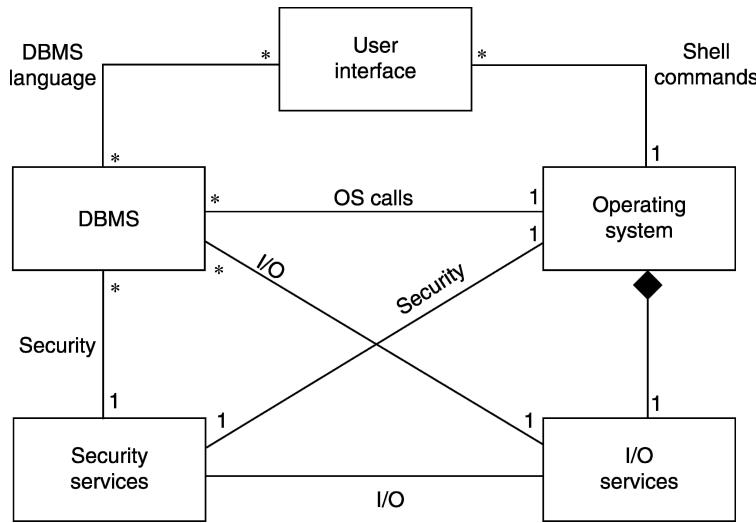
[15] presents an approach based on the high-level concepts and modeling activities of the secure Tropos methodology and enriched with low level security-engineering ontology and models derived from the UMLsec approach.

System Architecture for Security

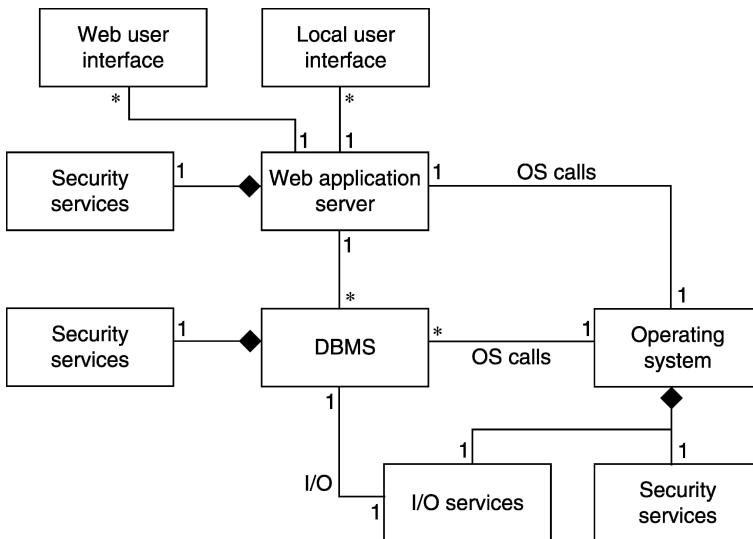
Whichever approach is used, there are basically three general architectural configurations to include security functions:



Secure Database Development. Figure 1. Standard placement of security services.



Secure Database Development. Figure 2. Common security services.



Secure Database Development. Figure 3. Architecture using a Web Application Server.

1. Figure 1 shows the standard approach. Here the DBMS and the operating system have their own set of security services.
2. Figure 2 shows a way to unify the design of the DBMS with the design of the OS, using an I/O and file subsystem and a security subsystem to be used by both the DBMS and the OS.
3. Figure 3 is an extension of the standard approach where a Web Application Server (WAS) unifies security for several databases. The WAS applies a common conceptual model to the information and can integrate different types of databases.

These configurations can be used in either of the approaches discussed earlier. Within each configuration it is possible to use security kernels and virtual machines.

Key Applications

Clearly, the first approach makes sense when the objective is a secure DBMS product, since it is not possible to know what user applications will be supported in the future. The only choice is then to build a system which is as secure as possible within these constraints and within a reasonable cost.

In the second case, the type of application to be supported is known. This gives the designers the flexibility of choosing an appropriate existing database system, as done in [9], or to build the DBMS to reach the required degree of security. If the complete DBMS is to be built, the first approach is appropriate, using as parameter the degree of security.

Cross-references

- ▶ [Access Control Administration Policies](#)
- ▶ [Access Control Policy Languages](#)
- ▶ [Architecture-Conscious Database System](#)
- ▶ [Application Server](#)
- ▶ [Authentication](#)
- ▶ [Authorization](#)
- ▶ [Concurrency Control – Traditional Approaches](#)
- ▶ [Data Stream Management Architectures and Prototypes](#)
- ▶ [Data Warehouse Life-Cycle and Design](#)
- ▶ [Data Warehouse Security](#)
- ▶ [Database Design](#)
- ▶ [Database Security](#)
- ▶ [DB Middleware](#)
- ▶ [DBMS](#)
- ▶ [Discretionary Access Control](#)
- ▶ [Distributed Database Design](#)
- ▶ [Distributed Database Systems](#)
- ▶ [Distributed DBMS](#)
- ▶ [Mandatory Access Control](#)
- ▶ [Metamodel](#)
- ▶ [Object Constraint Language](#)
- ▶ [Object Data Models](#)
- ▶ [Object-Role Modeling](#)
- ▶ [Privacy](#)
- ▶ [Process Life Cycle](#)
- ▶ [Process Structure of a DBMS](#)
- ▶ [Role Based Access Control](#)

Recommended Reading

1. Basin D.A., Doser J., and Lodderstedt T. Model driven security: from UML models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, 2006.
2. Bertino E. and Sandhu R. Database security – Concepts, approaches, and challenges. *IEEE Trans. Dependable Sec. Comput.*, 2(1):2–19, 2005.
3. Best B., Jürjens J., and Nuseibeh B. Model-based security engineering of distributed information systems using UMLsec. In Proc. 29th Int. Conf. on Software Eng., 2007, pp. 581–590.
4. Castano S., Fugini M., Martella G., and Samarati P. *Database Security*. Addison-Wesley, 1994.

5. Fernandez E.B., Gudes E., and Song H. A model for evaluation and administration of security in object-oriented databases. *IEEE Trans. Knowl. Database Eng.*, 6(2):275–292, 1994.
6. Fernandez E.B., Larrondo-Petrie M.M., Sorgente T., and VanHilst M. A methodology to develop secure systems using patterns, Chapter V. In *Integrating Security and Software Engineering: Advances and Future Vision*, H. Mouratidis, P. Giorgini (eds.). IDEA Press, 2006, pp. 107–126.
7. Fernandez E.B., Summers R.C., and Wood C. *Database Security and Integrity* (Systems Programming Series). Addison-Wesley, 1981.
8. Fernández-Medina E. and Piattini M. Extending OCL for secure database development. In Proc. Int. Conf. on the Unified Modeling Language, 2004, pp. 380–394.
9. Fernández-Medina E. and Piattini M. Designing secure databases. *Inf. Softw. Technol.*, 47(7):463–477, 2005.
10. Fugini M. Secure database development methodologies. In *Database Security: Status and Prospects*, C.E. Landwehr (ed.). Elsevier, 1987, pp. 103–129.
11. Ge X., Polack F., and Laleau R. Secure Databases: an Analysis of Clark-Wilson Model in a Database Environment. In Proc. 16th Int. Conf. on Advanced Information Systems Eng., 2004, pp. 234–247.
12. Hafner M. and Breu R. Towards a MOF/QVT-Based Domain Architecture for Model Driven Security. In Proc. 9th Int. Conf. Model Driven Eng. Lang. and Syst., 2006.
13. Jürjens J. *Secure Systems Development with UML*. Springer, New York, 2004.
14. Jürjens J. Sound methods and effective tools for model-based security engineering with UML. In Proc. 27th Int. Conf. on Software Eng., 2005, pp. 322–331.
15. Mouratidis H., Jürjens J., and Fox J. Towards a comprehensive framework for secure systems development. In Proc. 18th Int. Conf. on Advanced Information Systems Eng., 2006, pp. 48–62.

Secure Database Systems

- ▶ [Multilevel Secure Database Management Systems](#)

Secure Datawarehouses

- ▶ [Data Warehouse Security](#)

Secure DBMS Development

- ▶ [Secure Database Development](#)

Secure Hardware

► Trusted Hardware

Secure Multiparty Computation Methods

MURAT KANTARCIOĞLU¹, JAIDEEP VAIDYA²

¹University of Texas at Dallas, Richardson, TX, USA

²Rutgers University, Newark, NJ, USA

Definition

The problem of preserving privacy while allowing data analysis can be attacked in many ways. One way is to avoid disclosing data beyond its source while still constructing data mining models equivalent to those that would have been learned on an integrated data set. This follows the approach of Secure Multiparty Computation (SMC). SMC refers to the general problem of computing a given function securely over private inputs while revealing nothing extra to any party except what can be inferred (in polynomial time) from its input and output. Since one can prove that data are not disclosed beyond its original source, the opportunity for misuse is not increased by the process of data mining.

The definition of privacy followed in this line of research is conceptually simple: no site should learn anything new from the *process* of data mining. Specifically, anything learned during the data mining process must be derivable given one's own data and the final result. In other words, nothing is learned about any other site's data that is not inherently obvious from the data mining result. In the context of data mining, the approach followed in this research has been to select a type of data mining model to be learned and develop a protocol to learn the model while meeting this definition of privacy.

Historical Background

Privacy-preserving data mining can be defined as the problem of how to mine data when it is not possible to see it. Two seminal papers [1,9] first considered this problem and proposed different ways to attack it. Both looked at the problem of constructing decision trees from distributed data in a privacy-preserving manner. Agrawal and Srikant [1] proposed a randomization approach based on perturbing the input data and

reconstructing the distribution. Lindell and Pinkas [9] proposed a cryptographic solution based on secure multiparty computation. This entry describes the second approach following the application of secure multiparty computation methods to data mining.

Secure Multiparty Computation (SMC) originated with Yao's Millionaires' problem [15]. The basic problem is that two millionaires would like to know who is richer, with neither revealing their net worth. Abstractly, the problem is to simply compare two numbers, each held by one party, without either party revealing its number to the other. Yao [15] presented a generic circuit evaluation based solution for this problem as well as generalizing it to any efficiently computable function restricted to two parties. Goldreich et al. [6] generalized this to multi-party computation and proved that there exists a secure solution for any functionality. There has been significant theoretical work in this area. The restriction of polynomially time bounded passive adversaries has been removed. Similarly, work has been extended to active adversaries, as well as mobile adversaries. While much effort has been due to efficiency reasons, it is completely infeasible to directly apply the theoretical work from SMC to form secure protocols for privacy-preserving data mining.

Thus, work in privacy-preserving data mining has focused on creating specialized efficient solutions in the context of data mining. Starting with the work of Lindell and Pinkas [9], secure methods have been proposed for various tasks such as association rule mining [7,13], clustering [8], classification [2], and outlier detection [12]. [14] gives a good overview of much of this work.

Foundations

The basic ideas used in SMC based privacy-preserving data mining techniques are now illustrated using a commonly deployed public key encryption technique called homomorphic encryption [11]. More formally, let $E_{pk}(\cdot)$ denote the encryption function with public key pk and $D_{pr}(\cdot)$ denote the decryption function with private key pr . A secure public key cryptosystem is called additively homomorphic if it satisfies the following requirements: (i) Given the encryption of m_1 and m_2 , $E_{pk}(m_1)$ and $E_{pk}(m_2)$, there exists an efficient algorithm to compute the public key encryption of $m_1 + m_2$, denoted $E_{pk}(m_1 + m_2) := E_{pk}(m_1) + h E_{pk}(m_2)$. (ii) Given a constant k and the encryption of m_1 , $E_{pk}(m_1)$, there exists an efficient

algorithm to compute the public key encryption of km_1 , denoted $E_{pk}(km_1) := k \times_h E_{pk}(m_1)$.

Using the homomorphic encryption technique, one can easily develop many secure protocols. For example, consider the case where three sites S_1, S_2 and S_3 want to add their private values (resp.) v_1, v_2 , and v_3 to learn $v_1 + v_2 + v_3$ securely. A simple protocol for the above task using homomorphic encryption can be given as follows: S_1 creates a homomorphic encryption public and private key pair, and sends the public key to S_2 and S_3 . In addition, S_1 computes $e_1 = E_{pk}(v_1)$ and sends e_1 to S_2 . Using the homomorphic encryption scheme, S_2 can calculate $e_2 = e_1 +_h E_{pk}(v_2)$ and can send e_2 to S_3 . Similarly, S_3 can calculate $e_3 = e_2 +_h E_{pk}(v_3) = E_{pk}(v_1 + v_2) + E_{pk}(v_3)$. Finally, S_1 can decrypt the e_3 to compute $D_{pr}(e_3) = v_1 + v_2 + v_3$. If all the parties follow the protocol exactly, it can be shown that nobody learns anything other than the final result. The obvious question is what happens when the parties do not follow the protocol exactly. Clearly S_3 can collaborate with S_1 to learn the private value v_2 because if S_3 sends the message e_2 to S_1 then S_1 can compute $D_{pr}(e_2) - v_1 = (v_1 + v_2) - v_1 = v_2$ to learn v_2 .

The above example indicates that when considering privacy, one must first model the different adversarial behaviors that an attacker can assume. The SMC literature defines two basic adversarial models:

Semi-Honest: Semi-honest (or Honest but Curious) adversaries follow the protocol faithfully, but can try to infer the secret information of the other parties from the data they see during the execution of the protocol.

Malicious: Malicious adversaries may do anything to infer secret information. They can abort the protocol at any time, send spurious messages, spoof messages, collude with other (malicious) parties, etc.

While the semi-honest model may seem questionable for privacy (if a party can be trusted to follow the protocol, why would they not be trusted with the data?), it does meet several practical needs for early adoption of the technology. Consider the case where credit card companies jointly build data mining models for credit card fraud detection. In many cases the parties involved already have authorization to see the data (e.g., the theft of credit card information from CardSystems involved data that CardSystems was expected to see during processing). The problem is that *storing* the data brings with it a responsibility (and cost) of protecting that data; CardSystems was

supposed to delete the information once the processing was complete. If parties could develop the desired models without seeing the data, then they are saved the responsibility (and cost) of protecting it. Also the simplicity and efficiency possible with semi-honest protocols will help speed adoption so that trusted parties are saved the expense of protecting data other than their own. As the technology gains acceptance, malicious protocols will become viable for uses where the parties are not mutually trusted.

In either adversarial model, there exist formal definitions of privacy [5]. Informally, the definition of privacy is based on equivalence to having a trusted third party perform the computation. This is the gold standard of secure multiparty computation. Imagine that each of the data sources gives their input to a (hypothetical) trusted third party. This party, acting in complete isolation, computes the results and reveals them. After revealing the results, the trusted party forgets everything it has seen. A secure multiparty computation approximates this standard: no party learns more than it would in the trusted third party approach.

One fact is immediately obvious: no matter how secure the computation, some information about the inputs may be revealed. This is a result of the computed function itself. For example, if one party's net worth is \$100,000, and the other party is richer, one has a lower bound on their net worth. This is captured in the formal SMC definitions: any information that can be inferred from one's own data and the result can be revealed by the protocol. Thus, there are two kinds of information leaks; the information leak from the function computed irrespective of the process used to compute the function and the information leak from the specific process of computing the function. Whatever is leaked from the function itself is unavoidable as long as the function has to be computed. In secure computation, the second kind of leak is provably prevented. There is *no* information leak whatsoever due to the process.

While the generic secure multi-party computation methods exist, they pose significant computational problems. The challenge of privacy-preserving distributed data mining is to develop algorithms that have reasonable computation and communication costs on real-world problems, and prove their security with respect to the SMC definition. The typical approach taken is to reduce the large domain problem to a series of smaller sub-tasks and to use secure

cryptographic protocols to implement those smaller sub-tasks.

In the following, the common secure sub-protocols used in privacy-preserving distributed data mining are now described. As far as possible, for each sub-protocol, a version using only homomorphic encryption is described. Unless otherwise stated, all the sub-protocols are secure in the semi-honest model with no collusion, and all the arithmetic operations are defined in some large enough finite field.

Following the description of the subprotocols, it is shown how different algorithms could be implemented using these secure sub-protocols. Since these common building blocks are quite general, using the Composition theorem [5], they can be combined to create new privacy preserving algorithms in the future.

Secure Sum

Secure Sum securely calculates the sum of values from individual sites. As seen above, homomorphic encryption can easily be used to securely compute the sum local values. Assuming three or more parties and no collusion, a more efficient method can be found in [7].

Secure Comparison / Yao's Millionaire Problem

Assume that two sites, each having one value, want to compare the two values without revealing anything else other than the comparison result. Secure Comparison methods can be used to solve the above problem. To the best of our knowledge, secure circuit evaluation based approaches still provide the best performance [15].

Dot Product Protocol

Securely computing the dot product of two vectors is another important sub-protocol required in many privacy-preserving data mining tasks. Many secure dot product protocols have been proposed in the past. Among those proposed techniques, the method of Goethals et al. [4] is quite simple and provably secure. It is now briefly described.

The problem is defined as follows: Alice has a n -dimensional vector $\vec{X} = (x_1, \dots, x_n)$ while Bob has a n -dimensional vector $\vec{Y} = (y_1, \dots, y_n)$. At the end of the protocol, Alice should get $r_a = \vec{X} \cdot \vec{Y} + r_b$ where r_b is a random number chosen from uniform distribution that is known only to Bob, and $\vec{X} \cdot \vec{Y} = \sum_{i=1}^n x_i \cdot y_i$. The key idea behind the protocol is to use a homomorphic encryption system that can be used to perform

arithmetic operations over encrypted data. Using such a system, it is quite simple to build a dot product protocol. If Alice encrypts her vector and sends it in encrypted form to Bob, using the additive homomorphic property, Bob can compute the dot product. The specific details can be found in [4].

Oblivious Evaluation of Polynomials

Another important sub-protocol required in privacy-preserving data mining is the secure polynomial evaluation protocol. Consider the case where Alice has a polynomial P of degree k over some finite field \mathcal{F} . Bob has an element $x \in \mathcal{F}$ and also knows k . Alice would like to let Bob compute the value $P(x)$ in such a way that Alice does not learn x and Bob does not gain any additional information about P (except $P(x)$). This problem was first investigated by [10]. Subsequently, there have been more protocols improving the communication and computation efficiency as well as extending the problem to floating point numbers.

Privately Computing $\ln x$

For entropy measures used in data mining, one must be able to privately compute $\ln x$, where $x = x_1 + x_2$ with x_1 known to Alice and x_2 known to Bob. Thus, Alice should get y_1 and Bob should get y_2 such that $y_1 + y_2 = \ln x = \ln(x_1 + x_2)$. One of the key results presented in [9] was a cryptographic protocol for this computation. One point to note is that $\ln x$ is *Real* while general cryptographic tools work over finite fields. Therefore, $\ln x$ is actually multiplied with a known constant to make it integral. The basic idea behind computing random shares of $\ln(x_1 + x_2)$ is to use the Taylor approximation for $\ln x$. Thus, shares for the Taylor approximation are actually computed. The actual details of the protocol, as well as the proof of security, can be found in [9].

Secure Intersection

Secure Intersection methods are useful in data mining to find common rules, frequent itemsets etc., without revealing the owner of the item. Many algorithms have been developed for calculating Secure Set Intersection. For example, [13] provides an efficient solution. However, a secure set intersection protocol that utilizes secure polynomial evaluation [3] is described below. Let us assume that Alice has set $X = \{x_1, \dots, x_n\}$ and Bob has set $Y = \{y_1, \dots, y_n\}$. Our goal is to securely calculate $X \cap Y$. By representing set X as a polynomial and using

polynomial evaluation, Alice and Bob can calculate $X \cap Y$ securely.

Secure Set Union

Secure union methods are useful in data mining to allow each party to give its rules, decision trees etc. without revealing the owner of the item. Union of items can be easily evaluated using SMC methods if the domain of the items is small. Each party creates a binary vector (where the i th entry is 1 if the i th item is present locally). At this point, a simple circuit that *or's* the corresponding vectors can be built and securely evaluated using general secure multi-party circuit evaluation protocols. However, in data mining, the domain of the items are usually very large, potentially infinite. This problem can be overcome using approaches based on commutative encryption [7].

Key Applications

This section overviews how different sub-protocols described above could be used to create various privacy-preserving distributed data mining (PPDM) algorithms for different data models. In each of the discussed PPDM algorithms general data mining functionality is reduced to a computation of secure sub-protocols.

In the following discussion, horizontal partitioning of data implies that different sites collect the same set of information about different entities. Vertical data partitioning implies that different sites collect different features of information for the same set of entities. While this entry does not explicitly discuss arbitrary partitioning, the building blocks presented above are actually useful even in that case.

Classification

In the first work on privacy-preserving distributed data mining on horizontally partitioned data [9], the goal is to securely build an ID3 decision tree where the training set is horizontally distributed between two parties. The basic idea is that finding the attribute that maximizes information gain is equivalent to finding the attribute that minimizes the conditional entropy. The conditional entropy for an attribute for two parties can be written as a sum of the expression of the form $(v_1 + v_2) \times \log(v_1 + v_2)$. The authors use the secure log algorithm, secure polynomial evaluation, and secure comparison sub-protocols to securely calculate the expression $(v_1 + v_2) \times \log(v_1 + v_2)$ and

show how to use this function for building the ID3 securely. Correspondingly, decision trees for vertically partitioned data can also be built if the attribute with maximum entropy gain can be found. If the class attribute is present with all parties, this can be easily done. But even when the class attribute is only present with one of the parties, the secure scalar product protocol can be used to compute counts of transactions having certain attribute values and class values. This can then be used to compute the information gain for the attribute, and thus to decide the best attribute. Naïve Bayes classifiers can also be built for both horizontally and vertically partitioned data using combinations of the secure sum, secure scalar product and secure comparison primitives. [14] provides more details.

Association Rule Mining

The essential problem in association rule mining is the problem of finding frequent itemsets (meeting some support threshold). Once frequent itemsets are found, it is easy to find association rules meeting certain confidence thresholds. For horizontally partitioned data, [7] showed that for every candidate itemset, the support at each site can be computed locally. Now, a secure sum followed by a secure comparison is sufficient to evaluate if a candidate itemset is indeed frequent. However, this still requires the knowledge of the candidate itemsets. [7] uses some additional techniques (such as Secure Union) to ensure that candidate itemsets contributed by each site are also kept secret along with their support values. Similarly, [13] shows that the problem of finding frequent itemsets in vertically partitioned data can be reduced to the problem of securely computing the scalar product of multiple vectors (or equivalently as the problem of finding the size of the intersection set). Once this is done, finding globally valid association rules is quite simple.

Clustering

Several solutions for privacy-preserving clustering have been proposed. Lin et al. [8] propose a privacy preserving EM algorithm for secure clustering of horizontally partitioned data. EM clustering is an iterative algorithm. Each iteration consists of an expectation (E) step followed by a maximization (M) step. In the E-step, the expected value of the cluster membership for each entity is determined. In the M-step, the each cluster distribution parameters are re-estimated to maximize the likelihood of the data,

given the expected estimates of the membership. Lin et al. [8] show that computing the cluster parameters at each iteration can be easily done via secure summation once the total number of objects is known. Once computed, the cluster parameters are assumed to be public (i.e., known to all parties). Therefore each party can then locally assign its entities to the appropriate clusters. This is repeated until the algorithm converges or until a sufficient number of iterations have been carried out. Similar solutions exist for vertically partitioned data as well.

Outlier Detection

The goal of outlier detection is to find anomalies or outliers in the data. This requires a definition/metric of outlyingness. Many such metrics (with varying degrees of sophistication) have been defined in the statistical literature. One of the simplest metrics is that of $DB(p, d)$ outliers. Under this definition, an entity e in the dataset DB is said to be an outlier if more than p percentage of the entities in the dataset DB are farther than distance d from e . Thus, to figure out if an entity is an outlier, several tasks need to be performed: first, the distance of this entity to other entities must be computed; next, one must check if the number of farther entities is more than the given threshold. Vaidya and Clifton [12] show how to do this for both horizontally and vertically partitioned data. The key primitives used are the secure sum, secure comparison and secure dot product primitives. More detail can be found in [12].

Future Directions

Now that the key concepts behind secure multiparty computation methods have been presented, it is necessary to discuss some of the problems and challenges still open in this area. Inherently, the primary challenge with secure multiparty computation techniques lies with efficiency. Even with cryptographic accelerators and faster machines, since data mining is typically done over millions of transactions, this cost significantly balloons up. Even for other application areas, more efficient protocols are clearly needed. One alternative that has not been well explored is that of approximation. Instead of computing the exact results, it may make a lot more sense to compute approximations of the final results, especially if it gives huge efficiency improvements. This will be critical for development of real solutions in this area.

Cross-references

- ▶ [Horizontally Partitioned Data](#)
- ▶ [Privacy-Preserving Data Mining](#)
- ▶ [Vertically Partitioned Data](#)

Recommended Reading

1. Agrawal R. and Srikant R. Privacy-preserving data mining. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 439–450.
2. Du W. and Zhan Z. Building decision tree classifier on private data. In C. Clifton and V. Estivill-Castro (eds.). IEEE Int. Conf. on Data Mining Workshop on Privacy, Security, and Data Mining, 2002, pp. 1–8.
3. Freedman M.J., Nissim K., and Pinkas B. Efficient private matching and set intersection. In Proc. Int. Conf. Theory and Application of Cryptographic Techniques, 2004.
4. Goethals B., Laur S., Lipmaa H., and Mielikäinen T. On secure scalar product computation for privacy-preserving data mining. In Proc. the Seventh Annual Int. Conf. in Information Security and Cryptology, 2004, pp. 104–120.
5. Goldreich O. The Foundations of Cryptography, vol. 2, General Cryptographic Protocols. Cambridge University Press, London, 2004.
6. Goldreich O., Micali S., and Wigderson A. How to play any mental game – a completeness theorem for protocols with honest majority. In Proc. 19th ACM Symp. on the Theory of Computing, 1987, pp. 218–229.
7. Kantarcoğlu M. and Clifton C. Privacy-preserving distributed mining of association rules on horizontally partitioned data. IEEE Trans. Knowl. Data Eng., 16(9):1026–1037, 2004.
8. Lin X., Clifton C., and Zhu M. Privacy preserving clustering with distributed EM mixture modeling. Knowl. Inf. Syst., 8(1):68–81, 2005.
9. Lindell Y. and Pinkas B. Privacy preserving data mining. J. Cryptol., 15(3):177–206, 2002.
10. Naor M. and Pinkas B. Oblivious transfer and polynomial evaluation. In Proc. Thirty-First Annual ACM Symp. on Theory of Computing, 1999, pp. 245–254.
11. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. In Proc. Int. Conf. Theory and Application of Cryptographic Techniques, 1999, pp. 223–238.
12. Vaidya J. and Clifton C. Privacy-preserving outlier detection. In Proc. 2004 IEEE Int. Conf. on Data Mining, 2004, pp. 233–240.
13. Vaidya J. and Clifton C. Secure set intersection cardinality with application to association rule mining. J. Comput. Security, 13(4):593–622, November 2005.
14. Vaidya J., Clifton C., and Zhu M. Privacy-Preserving Data Mining, vol. 19 of Advances in Information Security, 1st edn. Springer, Berlin, 2005.
15. Yao A.C. How to generate and exchange secrets. In Proc. 27th IEEE Symp. on Foundations of Computer Science, 1986, pp. 162–167.

Secure Third-Party Data Management

- ▶ [Secure Data Outsourcing](#)

Secure Transaction Processing

INDRAKSHI RAY

Colorado State University, Fort Collins, CO, USA

Definition

Secure transaction processing refers to execution of transactions that cannot be exploited to cause security breaches.

Historical Background

Research in making transaction processing secure has progressed along different directions. Most of the early research in this area focused in processing multilevel transactions suitable for military applications. Such applications are characterized by having a set of security levels which are partially ordered using the dominance relation. The requirement is that information can flow from a dominated level to a dominating level but all other flows are considered to be illegal. The traditional concurrency control and recovery algorithms cannot be used for processing transactions in military applications because they cause illegal information flow. Most research in this area involved developing architectures, concurrency control and recovery mechanisms to prevent illegal information flow. Subsequently, researchers have also looked into the problem of processing real-time secure transactions. These transactions must satisfy real-time requirements together with preventing illegal information flow.

In the commercial sector, subsequent research focused on how to deal with the effect of malicious transactions that may have compromised the integrity of the database. The malicious transactions can damage one or more data items. Other transactions reading from these committed data items help spread the damage. Traditional recovery mechanisms cannot undo the effects of committed transactions. The research in this area focused on developing efficient techniques that will remove the effects of malicious transactions while minimizing the impact on good transactions.

Foundations

A comprehensive survey on the work in multilevel secure (MLS) databases was performed by Atluri et al. [11]. Some of the important results discussed in this paper are enumerated. An MLS database environment is characterized by a set of security levels L that are

partially ordered by the relation \prec . \prec is the dominance relation between classes and it is transitive, reflexive and anti-symmetric. For any two levels, $l_i, l_j \in L$, if level $l_i \prec l_j$, then l_j is said to dominate l_i . In this case, l_j and l_i are referred to as dominating and dominated level respectively. If neither $l_i \prec l_j$, nor $l_j \prec l_i$, then l_i and l_j are said to be incomparable. Each data object o in an MLS environment is associated with a security classification, denoted by $l(o)$, where $l(o) \in L$. Each user u is also cleared to some security level $l(u)$, where $l(u) \in L$. A user u cleared to security level $l(u)$ can log in at any security level l' , where l' is dominated by $l(u)$. All processes, including transactions, initiated during a session inherit the security level at which the user has logged in.

MLS systems allow information to flow from dominated to dominating levels but all other information flows are considered illegal. Direct information flow occurs by virtue of transactions reading and writing data items. When a transaction reads a data item, information flows from the data item to the transaction. Similarly, when a transaction writes a data item, information flows from the transaction to the data item. Such direct illegal information flow is prevented by using the simple security property and the \star -property of the Bell-LaPadula (BLP) model [12]. Simple security property states the condition under which a transaction can read a data item. A transaction T may read a data item O only if the security level of the data item, denoted by $L(O)$, is dominated by the security level of the transaction, denoted by $L(T)$, that is, only when $L(O) \prec L(T)$. During the read operation, information flows from the data item to the transaction. Thus, when $L(O) \prec L(T)$, information flows from dominated level to the dominating one. \star -property states the condition under which a transaction T can write a data item O . The write operation is allowed only when the security level of the data object, denoted by $L(O)$, is dominated by that of the transaction, denoted by $L(T)$. In other words, the write operation is allowed only when $L(T) \prec L(O)$. In this case the information flows from the dominated level to the dominating level. However, the \star -property does not prevent a transaction operating at the dominated level from corrupting data items at the dominating level. Thus, for reasons of integrity, a modified form of \star -property, known as restricted \star -property, is used in practice. The restricted \star -property allows a transaction T to write a data item O only if the security

levels of the transaction is the same as that of the object, that is, $L(T) = L(O)$. The properties stated in the BLP model are not adequate in preventing illegal information flows that occur through indirect means. One such example is the covert channels. A covert channel is an information flow mechanism within a system that is based on the use of system resources; it is not intended for communication between the regular users of the system. Unfortunately, the traditional transaction processing mechanisms can be exploited to establish a covert channel.

First, the discussion describes how the concurrency control protocols that are used in traditional transaction processing systems can be used to establish a covert channel. Two concurrency control mechanisms are considered: two-phase locking (2PL) and timestamp ordering (TO). 2PL requires transactions to acquire read lock (write lock) before reading (writing) a data item. A read lock on a data item can be acquired if no other transaction has a write lock on the same data item. A write lock can be acquired if no other transaction has any lock on the data item. The locks acquired by a transaction must eventually be released. Moreover, once a transaction releases a lock, it can no longer lock any other data item. TO requires each transaction t to have a unique timestamp $ts(t)$. Each data item x is associated with a read timestamp $rts(x)$ and a write timestamp $wts(x)$ that denotes the timestamp of the latest transaction that have read and written x respectively. The operations are executed on a first-come-first-serve basis. If the execution of an operation does not violate the serialization order specified by the timestamps of the transactions, it is executed. If not, the operation is not allowed and the transaction is aborted.

Suppose this MLS database system is associated with two security levels *low* and *high* where the level *high* dominates *low*. The transactions initiated by a *high* user can read all data items and write high data items in accordance with the security and the restricted- \star properties of the BLP model. The transactions initiated by a *low* user can read and write low data items only. Suppose there are two transactions T_h and T_l that have decided to collude and there are no other transactions executing in the system. The security level of T_h and T_l are high and low respectively. The database has a data item x whose security level is *low* - both transactions have decided to communicate by accessing this data item: T_h will read the data item x and T_l will write the data item. Assume that the concurrency control

mechanism uses two-phase locking (2PL). When T_h wants to read data item x , a read lock is placed on x which prohibits T_l from acquiring a write lock on x . Thus, T_h can selectively issue lock request on x to transmit information. T_l can measure the delay in acquiring lock and interpret the information. Thus, a covert communication channel has been established between T_l and T_h . A similar problem occurs if a timestamp-based protocol is used. Suppose $ts(T_l) < ts(T_h)$. If T_l attempts to write x after T_h has read it, the write operation is rejected and T_l is aborted. Here again, a high transaction can selectively cause a low transaction to abort and communicate information. Thus, concurrency control mechanisms in an MLS database must not only ensure serializability but must also eliminate such illegal information flows.

Researchers have proposed several concurrency control algorithms for processing transactions in an MLS database. The algorithms are dependent on the underlying architecture of the DBMS. First, the discussion focuses on the algorithms developed for the kernelized architecture. Some of the early solutions proposed are by Schaefer [35], Lamport [22], and Reed and Kanodia [34]. In these solutions, the transactions are allowed to proceed. However, before a transaction can commit it is validated. Thus, if a transaction at the dominating level has read some data item which has been updated by a transaction at the dominated level, the dominating level transaction must abort. Such algorithms will cause starvation of transactions at the dominating levels. Keefe and Tsai [20] proposed a protocol based on multiversion timestamp ordering. Although this protocol ensures serializable histories without causing starvation of transactions at the dominating levels, it has several problems. First, it requires a large number of versions to be maintained. Second, transactions at the dominating levels read stale data. Third, performance is an issue. Subsequent research [2,4,27] focused on limiting the number of versions to two. The idea is that transactions reading at the dominated level read from the snapshot, while those writing data at their own level write it on the current state.

Researchers have also proposed solutions that are suitable for replicated architectures. In such architectures, an MLS DBMS is constructed from several single-level DBMSs. The DBMS at level l will contain a copy of every data item that a transaction at level l can access. The first protocol for this architecture was proposed by Jajodia and Kogan [17]. The protocol

assumes that the set of security levels are totally ordered. Transactions are submitted to a global transaction manager (GTM) who is responsible for forwarding the transactions to the corresponding DBMSs. The updates made by the transactions must also be propagated to the DBMSs at the dominating levels. Two transactions that conflict at level l must be submitted at the dominating level l' in the order in which they commit. However, when transactions do not conflict, they are sent in an arbitrary order to the dominating levels. This may result in nonserializable histories as pointed out by Kang and Keefe [18]. Costich [14] improves upon the Jajodia-Kogan protocol in the following manner. First, it reduces the amount of trust required to implement GTM. Second, it does not require the security levels to form a total order. However, McDermott, Jajodia, and Sandhu [26], illustrate that certain security posets cause the protocol to deadlock and block update projections and produce non-serializable executions. Subsequently, researchers [3–5,18] have characterized the posets that create this problem. An example will help illustrate the problem. Suppose there are the following security levels: A , B , C , AB , BC , AC , and ABC . The dominance relationship between the levels is as follows: $A \prec AB$, $A \prec AC$, $B \prec AB$, $B \prec BC$, $C \prec AC$, $C \prec BC$, $AB \prec ABC$, $BC \prec ABC$, and $AC \prec ABC$. Three update transactions T_1 , T_2 and T_3 are submitted at levels A , B , and C respectively. These transactions execute at the DBMS at these levels and are then propagated to dominating levels. Thus, T_1 , T_2 must execute in the DBMS at level AB , T_2 , T_3 must execute at level BC , and T_3 , T_1 must execute at level AC . Suppose T_1 is serialized before T_2 in level AB , T_2 is serialized before T_3 in level BC , and T_3 is serialized before T_1 in level AC . Now these updates must be propagated to level ABC . The DBMS at level ABC must respect the serialization orders of the dominated levels. Here is a deadlock situation because the orders are conflicting. This problem is solved by ordering transactions according to the timestamps generated when the transactions commit for the first time and using a conservative TO protocol which ensures that update projections are never aborted.

The transactions discussed so far have a single security level associated with them and are termed single-level transactions. These transactions can read at multiple security levels but can update data at one security level only. The problem with single-level transactions is that they cannot preserve integrity

constraints spanning multiple security levels. Thus, for these databases serializability is an overly restrictive correctness criterion [16,25]. Jajodia and Atluri [16] have proposed weaker notions of correctness, such as, levelwise serializability, one-item read serializability, and pairwise serializability for MLS databases having single-level transactions. Other researchers have proposed the notion of multilevel transactions. A multilevel transaction is associated with a set of security levels. It allows the transaction to read and write data items at multiple security levels. A multilevel transaction is composed of a set of subtransactions. Each subtransaction is associated with a single security level and performs operations following the simple security and restricted- \star property of the BLP model. Ideally, a multilevel transaction must have the properties of atomicity, consistency, isolation, and durability and it also should not cause any illegal information flow by virtue of its execution. However, it is often not possible to guarantee atomicity without causing illegal information flow [13,36]. Towards this end, Blaustein et al. [13] defines varying degree of atomicity that can be achieved by multilevel transactions. Ray, Ammann, and Jajodia [31] provide a notion of semantic atomicity which is suitable for multilevel transactions. The application containing the transactions are formally analyzed to give assurance of the satisfaction of this property. This work does not use serializability as the correctness criterion but uses the notion of semantic correctness.

Protocols for distributed transaction also may have to be modified for multilevel secure databases. Consider, for instance, the early prepare protocol that ensures atomicity for distributed transactions. When a transaction T_i is submitted, the coordinator decomposes it into subtransactions, say, T_{ij} and T_{ik} which are distributed to the participants at sites j and k for processing. The participants execute the subtransactions and replies to the coordinator with a prepare/no vote. If all the sites have responded with a prepare vote, the transaction will be committed and the coordinator sends a commit message to all the participants. Now suppose that the transaction T_i is executing on a MLS database. Since T_i is a distributed transaction, it is possible that the subtransaction T_{ij} has finished its execution and entered the prepare state before T_{ik} completes. Some other subtransaction say T_{mj} at the dominated level may want to write a data item, say x , that has been read by T_{ij} . To prevent a covert channel,

the lock on x must be released by T_{ij} . Since T_{ik} is executing at a different site, it is possible that T_{ik} will acquire a lock after T_{ij} releases the lock. This will violate the two phase locking rule and may result in non-serializable executions.

This motivated Atluri, Bertino, and Jajodia [8] to propose a new protocol called secure early prepare (SEP). The coordinator decomposes the transaction T_i into subtransactions $T_{i1}, T_{i2}, \dots, T_{in}$ and sends them together with their security levels to the participants. The participants on completing their work successfully responds with a yes vote as before. However, if the participant has read a data item at the dominated level, it also sends a read-low indicator bit. If none of the participants have read low data items, the transaction proceeds like the early prepare protocol. However, if at least one participant has read a data item at the dominated level, additional rounds of message are necessary. In such cases, the coordinator sends a confirm message to all participants who have read data items at dominated levels. If the participant has not released any locks so far, it responds with a confirmed message. Otherwise it responds with a non-confirmed message. When the coordinator receives confirmed messages from all the participants who have read data items at dominated levels, the coordinator sends a commit message to all the participants. Otherwise, an abort message is sent. In a subsequent work [8], the authors propose an optimization to SEP that avoids some unnecessary aborts caused by SEP and also reduces the number of messages. Ray et al. [32] also attempts to improve upon SEP – instead of aborting subtransactions that have read from low data items, it rolls back the subtransaction to an earlier savepoint and reexecutes it. On successful reexecution, the participant sends a yes message. When all subtransactions have responded with a yes message, the transaction is committed. Otherwise, it is aborted.

Some researchers [1, 19, 37] have also looked into secure real-time transaction processing. The goal in such systems is to prevent illegal information flow as well maintain the timing constraints required for real-time applications. When both security constraints and real-time constraints cannot be satisfied, some works [1, 37] trade-off security in order to improve the performance. Others [15, 19] do not compromise security for the sake of performance. George and Haritsa [15] propose a concurrency control mechanism in which data conflicts are resolved in favor of the dominated level. Within a given level, data conflicts are resolved in

favor of the earliest transaction deadline. Kang et al. [19] improve upon the work presented by George and Haritsa [15] by providing guarantees on average/transient miss ratios. A separate work [30] describes a new concurrency control protocol, known as, multiversion locking protocol with freezing, for processing secure real-time transactions.

Researchers have also investigated the impact of multilevel security on extended transaction models, such as workflows. A workflow is characterized as having a set of tasks and dependencies specified between the tasks. In an MLS workflow, each task is associated with a single security level and is allowed to read and write data items provided they obey the BLP rules. However, the dependencies between tasks at different levels may cause illegal information flow. Towards this end, Atluri et al. [10] have proposed an approach that redesigns the dependencies such that illegal information flow does not occur. A separate work [9] argues about how mandatory and discretionary access controls can be enforced in a workflow.

Secure transaction processing in non-MLS database systems focused on survivability. Attacks will occur in spite of sophisticated prevention mechanisms. The issue is how to identify the attack, confine it, assess the damage caused by it, and repair the damage in a timely manner. One of the early works in damage detection and recovery is by Ammann et al. [7]. After an attack occurs, the data items are marked with different colors to indicate the severity of the damaged. The authors define a notion of consistency for databases in which some data may have been damaged. Clean data must satisfy the integrity constraints defined over them. Damaged data must satisfy a set of relaxed integrity constraints. The authors classify transactions into three categories: *attack transactions*, *normal transactions* and *countermeasure transactions*. Attack transactions damage data items. Normal transactions sometimes help spread the damage. The normal transaction access protocol determines how the damage is propagated and ensures that the database satisfy the consistency constraints. The countermeasure transactions detect and repair the effects of an attack. These transactions must execute as trusted processes. Detection transactions change the marking of data items whereas repair transactions alter the value of data items. When an attack has occurred, the state of the database before the execution of attack transactions can be retrieved using snapshots. Towards this end, the paper proposes a

technique by which snapshots can be generated while the database is servicing normal transactions.

Survivability issues and repair from malicious attacks have received attention in the database context. Ammann et al. [6] propose repair algorithms for traditional database systems that help to recover from the damage caused by malicious transactions. A two pass static algorithm is proposed where the first pass scans the log forward to locate all malicious and suspect tasks and the second pass goes backward from the end of the log to undo all malicious and suspect tasks. They also proposed a dynamic repair algorithm that continues to accept new transactions while repair is taking place. Panda et al. [21] have also proposed a number of algorithms on damage assessment and repair; some of these store the dependency information in separate structures so that the log does not have to be traversed for damage assessment and repair. Ray et al. [33] improve upon the time taken to assess the damage by using a dependency graph to store the dependencies and using depth-first search to retrieve the affected transactions. Liu and Jajodia [24] present a multi-phase damage confinement model. In the initial confinement phase, an estimation is done with respect to the damaged items. The estimation may not be accurate. The authors propose several schemes about damage confinement. The first one maintains timestamps. The initial confinement confines all data items that were updated after the commitment of the bad transactions. A damage assessor unconfines data items that are written by transactions not dependent on the bad transactions. This simple scheme causes damage leakage because a data item unconfined by an unaffected transaction can be updated by an affected transaction. Thus, even temporarily releasing this data item can cause damage spreading. The second scheme takes care of this problem. Confining the data items and later unconfining them usually takes some time. To reduce the relaxation latency, the authors propose a third scheme which uses transaction access patterns to unconfine data items that were not affected by the bad transaction. Panda and Giordano [28] provide two techniques for performing damage assessment and recovery. The first algorithm does detection and recovery simultaneously and is not very efficient. Moreover, new transactions are blocked until recovery is complete. These two shortcomings are removed in the second algorithm. Most of the work on damage assessment are based on transaction dependency approach. In these approaches, the goal is to

identify affected transactions which must be undone and then re-executed. However, this may involve unnecessary undoing and redoing of operations. This is because not all operations of an affected transaction are influenced by a bad transaction. Towards this end, Panda and Haque [29] propose a damage assessment technique based on data dependency approach – only the affected operations are undone and redone.

Damage assessment in distributed databases has also been studied by several researchers. Liu and Hao [23] propose a damage assessment technique for distributed database that incurs a high communication overhead. Zuo and Panda [40] propose two approaches for damage assessment in distributed databases. The first one is a peer-to-peer approach which does not require a coordinator to perform damage assessment. When a site knows that it has some global affected transactions, it sends a multicast message to other sites which were involved with these global transactions. The other sites on receiving this message may identify some more global affected transactions which are then broadcast. The process continues until no more new global affected transactions are detected. This approach incurs high communication overhead. The other approach requires a coordinator. There are three variants of this other approach. In the first one, known as receive and forward, the coordinator keeps information about all global transactions. Each site manager sends a list of global affected transaction identifiers to the coordinator. The coordinator informs the other sites of these affected transactions. The other sites check whether or not any new transactions are affected. If not, a clear message is sent to the coordinator. Otherwise, the identity of the affected transactions are sent. The process stops when all the other sites send a clear message. The second coordinator-based approach incurs less communication overhead. When a malicious transaction is identified, the coordinator requests other sites for their local dependency graphs. The coordinator builds a global dependency graph using this information. The global graph is used for identifying affected transactions. The third coordinator-based approach relies on sites sending their graphs periodically to the coordinator. In this approach, the local graphs are not merged.

Yu, Liu and Zang [38] describe an algorithm for on-line attack recovery of workflows. The algorithm tries to build the list of redo and undo tasks, after an independent Intrusion Detection System reports malicious tasks. They also relax the restriction

of executing order that exist in an attack recovery system; they introduced multi-version data objects to reduce unnecessary blocks in order to reduce degradation of performance in recovery. However, like in most existing papers, the authors only pay attention to restore consistency for data objects, while they do not analyze the correct actions needed in repair for different control-flow dependencies. In repair for advanced transactions, one needs to ensure that constraints of all control-flow dependencies are satisfied, and one should not treat all types of control-flow dependencies in the same manner – need to distinguish different types of control-flow dependencies and adopt different treatment to enforce these dependencies during repair. This issue is addressed in a subsequent work [39].

Key Applications

Critical information, such as health records, financial records are stored in the databases of an organization. Organizations must also interact with each other and share critical information to accomplish a particular mission. Security and privacy breaches involving such critical information have disastrous consequences. Thus, there is a need to formalize the concept of secure information flow both in the context of military and commercial databases. Ideally, mechanisms that enforce the secure information flow policies are needed. Automated capabilities that will track information flow and detect and thwart illegal flows are also needed. Since it is impossible to protect against all kinds of security and privacy breaches, it is also necessary to design systems that can automate to the extent possible the detection and repair from an attack.

Cross-references

- ▶ [Data Confidentiality](#)
- ▶ [Data Integrity Services](#)
- ▶ [Information Flow](#)
- ▶ [Mandatory Access Control](#)
- ▶ [Multilevel Secure Database Management System](#)
- ▶ [Transaction Processing](#)

Recommended Reading

1. Ahmed Q. and Vrbsky S. Maintaining security in firm real-time database systems. In Proc. 14th Annual Computer Security Applications Conference, 1998.
2. Ammann P., Jaekle F., and Jajodia S. A two-snapshot algorithm for concurrency control in secure multi-level databases. In Proc. IEEE Symp. on Security and Privacy, 1992, pp. 204–215.

3. Ammann P. and Jajodia S. Distributed timestamp generation in planar lattice networks. ACM Trans. Comput. Syst., 11(3): 205–225, 1993.
4. Ammann P. and Jajodia S. An efficient multiversion algorithm for secure servicing of transaction reads. In Proc. First ACM Conf. on Computer and Communication Security, 1994, pp. 118–125.
5. Ammann P., Jajodia S., and Frankl P. Globally consistent event ordering in one-directional distributed environments. IEEE Trans. Parallel Distrib. Syst., 7(6):665–670, 1996.
6. Ammann P., Jajodia S., and Liu P. Recovery from malicious transactions. IEEE Trans. Knowl. Data Eng., 14:1167–1185, 2002.
7. Ammann P., Jajodia S., McCollum C., and Blaustein B. Surviving information warfare attacks on databases. In Proc. IEEE Symp. on Security and Privacy, 1997.
8. Atluri V., Bertino E., and Jajodia S. Degrees of isolation, concurrency control protocols, and commit protocols. In Proc. IFIP WG11.3 Working Conf. on Database Security, 1995, pp. 259–274.
9. Atluri V. and Huang W.K. Enforcing mandatory and discretionary security in workflow management systems. J. Comput. Secur., 5(4):303–340, 1997.
10. Atluri V., Huang W.K., and Bertino E. A semantic-based execution model for multilevel secure workflows. J. Comput. Secur., 8(1):3–41, 2000.
11. Atluri V., Jajodia S., Keefe T.F., McCollum C., and Mukkamala R. Mutilevel secure transaction processing: status and prospects. In Proc. 10th IFIP WG11.3 Working Conf. on Database Security, 1996.
12. Bell D.E. and LaPadula L.J. Secure computer system: unified exposition and multics interpretation. Tech. Rep. MTR-2997, MITRE Corporation, 1975.
13. Blaustein B.T., Jajodia S., McCollum C.D., and Notargiacomo L. A model of atomicity for multilevel transactions. In Proc. IEEE Symp. on Research in Security and Privacy, 1993, pp. 120–134.
14. Costich O. Transaction processing using an untrusted scheduler in a multilevel database with replicated architecture. In Proc. IFIP WG11.3 Working Conf. on Database Security, 1992, pp. 173–190.
15. George B. and Haritsa J. Secure concurrency control in firm real-time databases. Distrib. Parallel Databases, 5:275–320, 1997.
16. Jajodia S. and Atluri V. Alternative correctness criteria for concurrent execution of transactions in multilevel secure databases. In Proc. IEEE Symp. on Security and Privacy, 1992, pp. 216–224.
17. Jajodia S. and Kogan B. Integrating an object-oriented data model with multilevel security. In Proc. IEEE Symp. on Security and Privacy, 1990, pp. 76–85.
18. Kang I. and Keefe T. Transaction management for multilevel secure replicated databases. J. Comput. Secur., 3:115–145, 1995.
19. Kang K., Son S., and Stankovic J. STAR: secure real-time transaction processing with timeliness guarantees. In Proc. 23rd IEEE Real-Time Systems Symp., 2002.
20. Keefe T. and Tsai W. Multiversion concurrency control for multilevel secure databases. In Proc. IEEE Symp. on Security and Privacy, 1990, pp. 369–383.

21. Lala C. and Panda B. Evaluating damage from cyber attacks: a model and analysis. *IEEE Trans. Syst. Man Cybern. A*, 31(4): 300–310, 2001.
22. Lamport L. Concurrent reading and writing. *Commun. ACM*, 20(11):806–811, 1977.
23. Liu P. and Hao X. Efficient damage assessment and repair in resilient distributed database systems. In Proc. 15th IFIP WG11.3 Working Conf. on Data and Application Security, 2001, pp. 75–89.
24. Liu P. and Jajodia S. Multi-phase damage confinement in database systems for Intrusion Tolerance. In Proc. 14th IEEE Computer Security Foundations Workshop, 2001.
25. Maimone W. and Greenberg I. Single-level multiversion schedulers for multilevel secure database systems. In Proc. 6th Annual Computer Security Applications Conf., 1990, pp. 137–147.
26. McDermott J., Jajodia S., and Sandhu R. A single-level scheduler for replicated architecture for multilevel secure databases. In Proc. 7th Annual Computer Security Applications Conf., 1991, pp. 2–11.
27. Pal S. A locking protocol for multilevel secure databases providing support for long transactions. In Proc. 10th IFIP WG11.3 Working Conf. on Database Security, 1996, pp. 183–198.
28. Panda B. and Giordano J. Reconstructing the database after electronic attacks. In Proc. 12th IFIP WG11.3 Int. Working Conf. on Database Security, 1998.
29. Panda B. and Haque K.A. Extended data dependency approach: a robust way of rebuilding database. In Proc. 2002 ACM Symp. on Applied Computing, 2002.
30. Park C., Park S., and Son S. Multiversion locking protocol with freezing for secure real-time database systems. *IEEE Trans. Knowl. Data Eng.*, 14(5):1141–1154, 2002.
31. Ray I., Ammann P., and Jajodia S. A semantic-based model for multi-level transactions. *J. Comput. Secur.*, 6(3):181–217, 1998.
32. Ray I., Bertino E., Jajodia S., and Mancini L. An advanced commit protocol for MLS distributed database systems. In Proc. 3rd ACM Conf. on Computer and Communications Security, 1996, pp. 119–128.
33. Ray I., McConnell R., Lunacek M., and Kumar V. Reducing damage assessment latency in survivable databases. In Proc. 21st British National Conf. on Databases, 2004.
34. Reed D. and Kanodia R. Synchronizations with event counts and sequencers. *Commun. ACM*, 22(5):115–123, 1979.
35. Schaefer M. Quasi-synchronization of readers and writers in a multi-level environment. Tech. Rep. TM-5407/003, System Development Corporation, 1974.
36. Smith K.P., Blaustein B.T., Jajodia S., and Notargiacomo L. Correctness criteria for multilevel secure transactions. *IEEE Trans. Knowl. Data Eng.*, 8(1):32–45, 1996.
37. Son S., Mukkamala R., and David R. Integrating security and real-time requirements using covert channel capacity. *IEEE Trans. Knowl. Data Eng.*, 12(6):865–879, 2000.
38. Yu M., Liu P., and Zang W. Multi-version attack recovery for workflow systems. In Proc. 19th Annual Computer Security Applications Conf., 2003, pp. 142–151.
39. Zhu Y., Xin T., and Ray I. Recovering from malicious attacks in workflow systems. In Proc. 16th Int. Conf. Database and Expert Syst. Appl., 2005.
40. Zuo Y. and Panda B. Damage discovery in distributed database systems. In Proc. 18th IFIP WG11.3 Working Conf. on Data and Applications Security, 2004.

Security Services

ATHENA VAKALI

Aristotle University, Thessaloniki, Greece

Synonyms

[Authentication](#); [Data confidentiality](#); [Data integrity services](#)

Definition

Given a set of local or distributed resources to be protected, a security service is a task (or set of tasks) that coherently performs processing or communication on behalf of the underlying system infrastructure, in order to support and employ several security requirements of both the system and the data sources. Such requirements involve authentication, PKI accessing etc. over the underlying resources. Security services typically implement portions of security policies and are implemented via particular processes which are called security mechanisms.

Key Points

Security services are well documented and described in X.800 documentation [1] for almost all the layers from the physical up to the application layer, whereas focus on security services applied on the Web is given in [2]. Physical and data layer security services focus on support confidentiality at various levels, namely at the connection, the traffic flow (both full and limited) and they are designed for peer to peer or multi-peer communications. At the network and transport layers security services involve authentication, access control, traffic flow, confidentiality and they are provided together or separately.

At the application layer, from X.800 the core security services are identified and they may be supported either singly or in combination, to employ access control for enforcing authentication, data confidentiality,

data integrity and non-repudiation. As highlighted in [2], these involve:

1. Services for subjects/clients and resources identities: to verify the identity of the subject who requests a source access and prevent malicious client attempts. Therefore, such services may be either authentication services (to verify an identity claimed by/for an entity) or Nonrepudiation services (to prevent either sender or receiver from denying a transmitted message).
2. Services for subjects/clients authorizations over resources: to manage relationships among clients and protected resources. Therefore, such services involve either access control services (for protecting system resources against unauthorized access) or data privacy (for protecting data against unauthorized disclosure-data confidentiality and changes-data integrity).

Cross-references

- ▶ Authentication
- ▶ Database Security
- ▶ Secure Database Development

Recommended Reading

1. Recommendation X.800, Security architecture for open systems, Interconnection for CCITT applications. <http://fag.grm.hia.no/IKT7000/litteratur/paper/x800.pdf>
2. Stoupa K. and Vakali A. Policies for Web Security Services, Chapter III. In Web and Information Security, E. Ferrari, B. Thuraisingham (eds.). Idea Group, USA, 2006.

Segmentation

- ▶ Cluster and Distance Measure

Selection

CRISTINA SIRANGELO
University of Edinburgh, Edinburgh, UK

Definition

Given a relation instance R over set of attributes U and a condition F , the selection $\sigma_F(R)$ returns a new

relation over U consisting of the set of tuples of R which satisfy F . The condition F is an atom of the form $A = B$ or $A = c$, where A and B are attributes in U , and c is a constant value.

The generalized selection allows more complex conditions: F can be an arbitrary boolean combination of atoms of the form $A = B$ or $A \neq B$ or $A = c$ or $A \neq c$. Moreover, if a total order is defined on the domain of attributes, more general comparison atoms of the form $A \alpha B$ or $A \alpha c$ are allowed, where α ranges over $\{=, \neq, <, >, \leq, \geq\}$.

Key Points

The selection is one of the basic operators of the relational algebra. It operates by “selecting” rows of the input relation. A tuple t over U satisfies the condition $A = B$ if the values of attributes A and B in t are equal. Similarly t satisfies the condition $A = c$ if the value of attribute A in t is c . Satisfaction of generalized selection atoms is defined analogously.

As an example, consider a relation $Exams$ over attributes $(course-number, student-number, grade)$, containing tuples $\{(EH1, 1001, A), (EH1, 1002, A), (GH5, 1001, C)\}$. Then $\sigma_{grade=A \wedge course-number = EH1}(Exams)$ is a relation over attributes $(course-number, student-number, grade)$ with tuples $\{(EH1, 1001, A), (EH1, 1002, A)\}$.

In the case that a relation schema is only specified by a relation name and arity, the result of the selection is a new relation having the same arity as the input one, containing the tuples which satisfy the selection condition. In this case the selection atoms are expressions of the form $j = k$ or $j = c$ (or $j \alpha k$ and $j \alpha c$ in the generalized selection). Here j and k are positive integers bounded by the arity of the input relation, identifying its j -th and k -th attribute, respectively.

Cross-references

- ▶ Relation
- ▶ Relational Algebra

Selective XML Dissemination

- ▶ XML Publish/Subscribe

Selectivity Estimation

EVAGGELIA PITOURA

University of Ioannina, Ioannina, Greece

Synonyms

[Selectivity estimation](#); [Cost estimation](#)

Definition

For each query, there are many equivalent execution plans. To choose the most efficient among these different query plans, the optimizer has to estimate their cost. Computing the precise cost of each plan is usually not possible without actually evaluating the plan. Thus, instead, optimizers use statistical information stored in the DBMS, such as the size of relations and the depth of the indexes, to estimate the cost of each plan. For large databases, this cost is dominated by the number of disk accesses.

Since, in general, the cost of each operator depends on the size of its input relations, it is important to provide good estimations of their selectivity, that is, of their result size.

Key Points

During query optimization, the optimizer enumerates potential execution plans for each query and evaluates their cost in order to choose the less expensive among them. In general, computing the exact cost of each query plan is not possible without actually evaluating the plan. Instead, optimizers make use of statistical information stored in the DBMS catalog to estimate the cost of each plan. Such statistics include the number of tuples in each relation, the size of each tuple and the number of distinct values that appear in each relation. The cost of a plan can be measured in terms of different resources, such as CPU, I/O, buffer utilization, and, in the case of parallel and distributed databases, communication costs. However, in large databases, the cost is usually dominated by disk accesses.

The cost of each plan is computed by combining the estimations of the cost of each of the operators appearing in the plan. Since the cost of an operator depends mainly on the sizes of its input relations, it is central to have good estimates of the result size of each operator that is going to be used as input to the next operator in the plan.

Take, for example, a selection condition consisting of a number of predicates. Each predicate has a reduction factor, which is the relative reduction in the number of result tuples caused by this predicate. There are many heuristic formulas for the reduction factors of different predicates. In general, they depend on the assumption of uniform distribution of values and independence among the various relation fields. More accurate reduction factors can be achieved by maintaining more accurate statistics, for example in the form of histograms or multidimensional histograms. The accuracy of the estimates also depends on how frequently the available statistics are updated to reflect the current database state.

Cross-references

- ▶ [Evaluation of Relational Operators](#)
- ▶ [Query Optimization](#)
- ▶ [Query Plan](#)

Recommended Reading

1. Chaudhuri S. An overview of query optimization in relational systems. In Proc. 17th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1998, pp. 34–43.
2. Ramakrishnan R. and Gehrke J. Database Management Systems. McGraw-Hill, New York, 2003.
3. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.

Selectivity for Predictive Spatio-Temporal Queries

- ▶ [Spatio-Temporal Selectivity Estimation](#)

Self-Maintenance of Views

HIMANSHU GUPTA

Stony Brook University, Stony Brook, NY, USA

Definition

A data warehouse is a collection of materialized views derived from base relations that may not reside at the warehouse. It is important to keep the views up to date in response to changes to the base relations. *Self-maintenance* of views involves maintaining the views,

using information that is strictly local to the warehouse: the view definitions and the view contents. Such self-maintenance of views (whenever possible) is more efficient than incremental maintenance or recomputation of views.

Key Points

In a data warehouse, views are computed and stored in the database to allow efficient querying and analysis of the data. These views stored at the data warehouse are known as *materialized views*, and are defined in terms of the base relations residing in data sources that may or may not be local to the warehouse. To keep the views consistent with the base data, any change reported by the data sources must be reflected in the views. In response to the changes at the base relations, the view can be either recomputed from scratch, or incrementally maintained, or self-maintained.

Incremental maintenance of a view involves propagating the changes at the source onto the view so that the view reflects the changes. While maintaining these views incrementally is often significantly more efficient than recomputing them from scratch (as done in most current data warehouses), it can still be expensive. For instance, in response to an update to a base relation, incremental maintenance of views defined as a join may involve looking up the non-updated base relations, which may reside in external sources. Some of these base relations may even be unavailable when the view needs to be maintained. Further, since base relations are independently updated, they may be read in an inconsistent state, often resulting in erroneous view updates. Thus, in data warehousing environments where maintenance is performed locally at the warehouse, an important incremental view-maintenance issue is how to minimize external base data access.

Motivated by the above, another approach of maintaining a view is called *self-maintenance*. In view self-maintenance, a view is maintained using information only local to the datawarehouse, viz., the view definition and the view contents. In general, a view may not be self-maintainable. However, in certain cases, a view can still be maintained using only a specified subset of the base relations. With the above approach of self-maintenance, it is possible to minimize the cost to maintain the data warehouse, shorten the time window during which the warehouse is inconsistent with the updated data sources, and avoid view update anomalies due to asynchronous base data updates.

There can be two notions [2] of self-maintainability (SM): the *compile-time SM* where a view is self-maintainable independently of the view's contents and the contents of the base relations, and under all updates of a certain type. The *runtime SM* is when a specific view is self-maintainable under a specific update and given the contents of a specified subset of base relations. Note that the runtime approach is more aggressive in the way that it may succeed in maintaining a view when the compile-time approach may fail. Below, works on run-time self-maintainability approach are described.

An important question is to determine whether a view is (run-time) self-maintainable. In other words, test for self-maintainability requires whether a unique new state is guaranteed, given an update to the base relations, an instance of the views, and an instance of a subset of the base relations. A second question is how to bring the view up to date using only the given information. Together, these two questions define the *view self-maintenance problem*.

The authors in [1,5] give self-maintainability conditions for views that are select-project-join queries with no self-joins and for single-relation insertions or deletions. Huyn in [3] solved the problem more efficiently than [1] for select-project-join views with no self-joins for single insertions. In particular, Huyn generate SQL queries that test whether a view is self-maintainable and update the view if it is. More specifically, he shows that for insertion updates and conjunctive queries: (i) the self-maintainability test are extremely simple queries that look for certain tuples in the view to be maintained, (ii) these tests can be generated from just the view definition using a simple algorithm based on the concept of “Minimum Z-Partition,” and (iii) view self-maintenance can also be expressed as simple update query over the view.

In a follow-up work [4], Huyn considers the view self-maintenance problem in the presence of multiple views, under arbitrary mixes of insertions and deletions. In particular, for conjunctive queries, he gives an algorithm that generates, at view definition time, the query expressions required to maintain the view in response to base updates. He generalizes his techniques to the problem of generalized self-maintenance problem, where in addition to the warehouse views, one is also given access to some of the base relations. In general, he provides better insight into the problem by showing that view

self-maintainability can be reduced to the problem of deciding query containment.

Cross-references

- ▶ [Data Warehouse](#)
- ▶ [View Maintenance](#)
- ▶ [Views](#)

Recommended Reading

1. Gupta A. and Blakeley J.A. Using partial information to update materialized views. *Inf. Syst.*, 20(9):641–662, 1995.
2. Gupta A. and Mumick I.S. Maintenance of materialized views: problems, techniques, and applications. *IEEE Data Eng. Bull.*, 18(2):3–18, 1995.
3. Huyn N. Efficient view self-maintenance. In Proc. Workshop on Materialized Views (VIEWS), 1996.
4. Huyn N. Multiple-view self-maintenance in data warehousing environments. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997.
5. Tompa F.W. and Blakeley J.A. Maintaining materialized views without accessing base data. *Inf. Syst.*, 13(4):393–406, 1988.

management, system-health monitoring, failure diagnosis and root-cause identification, configuration of backup procedures and other self-healing capabilities. The self-managing capabilities can be incorporated in a system using a number of architectural options. For example, such capabilities can be either built into the system itself and integrated with its normal functionality, or provided through external tools that for the database engine. The latter approach is often referred to as DB advisors, DB assistants, or DB wizards.

The notion of self-management for database systems comprises a wide spectrum of issues, and it is tempting to consider the partial automation of all database-related human activity as facets of self-management, for example, information integration. However, this entry defines self-management technology to be confined to system issues that arise with the operation of a database engine, thus excluding the tasks that do not directly affect the engine's operation.

Historical Background

Needs for tuning tools and certain self-managing capabilities have been around for decades. For example, analytic models for capacity planning has a long tradition in the mainframe world [13], and methods for incremental online reorganization of storage and indexing systems can be seen as early forms of self-management. Selected issues of automatic tuning have been addressed already in the 1980's, most notably, on index selection [11].

In the late 1980's and throughout the 1990's, both database system functionality and workloads became much richer and increased the complexity of system management. Together with the proliferation of database systems across a wide spectrum of IT applications, this created a shortage of sufficiently skilled system administrators and tuning experts. At the same time hardware and software-licensing costs were rapidly decreasing, so that human staff for system management became the key factor in total cost of ownership (TCO). These trends alerted the database-system industry in the mid 1990's and led to intensive research and development initiatives at all of the major system vendors.

Early work on more comprehensive strategies and principles of automatic tuning included the Comfort project at ETH Zurich [18,19] and the “DBMS Auto-pilot” work at the University of Wisconsin [2]. These projects were in turn inspired by prior work towards

Self-Management Technology in Databases

SURAJIT CHAUDHURI¹, GERHARD WEIKUM²

¹Microsoft Corporation, Redmond, WA, USA

²Max-Planck Institute for Informatics, Saarbruecken, Germany

Synonyms

[Self-managing database systems](#); [Autonomic database systems](#); [Self-tuning database systems](#); [Auto-administration and auto-tuning of database systems](#)

Definition

The total cost of ownership (TCO) for a database-centric information system is dominated by the expenses for highly skilled human staff in order to deploy, configure, administer, monitor, and tune the database system. Self-management technology for databases aims to automate these tasks to the largest possible extent and throughout the entire life-cycle of the information system. This involves many dimensions that determine the system performance and availability such as: workload analysis, capacity planning, physical database design, database statistics management for query optimization, load control, memory

adaptive and self-tuning operating systems [16], and particularly focused on resource control for dynamically evolving, mixed workloads. Starting in the mid 1990's interest in self-management was revived by the AutoAdmin project, which then became the leading initiative in this area [5,8,9]. AutoAdmin initially focused on physical database design, but subsequently also considered many other issues such as adaptive statistics management, system monitoring, and online tuning techniques. Good overviews of the progress achieved in the past ten years, in terms of both research contributions and product impact, are given by [1,8,9].

Foundations

General Framework

The overriding goal of self-management is to operate the database system at a satisfactory level of performance and service quality – at every point in time regardless of load peaks or shifts in workload characteristics. Specifically, the system should automatically adjust its configuration to evolving workloads. To this end, the key issue is to understand for each component of the system the dependencies that relate the system configuration and workload properties to the resulting performance measures:

$$\text{configuration} \times \text{workload} \rightarrow \text{performance}$$

All three parameters of this relation need to be interpreted broadly:

- *System configuration* includes the *hardware setup* (number and speed of processor cores, memory size, number and characteristics of disks and access channels, etc.), the *software setup* at system-boot time (e.g., thresholds for lock escalation or memory-pressure handling), the *physical database design* (indexes, materialized views, etc.), the *backup or replication procedures* and their parameters (e.g., frequency and granularity of backups, consistency protocols for replication), and also the system's *run-time adaptation policies* to handle newly arising conditions (e.g., the scheduling policy and workload-class priorities, the memory allocation policy, etc.) as well as the system's *exception-handling policies* (e.g., load shedding under memory pressure).
- *Workload characteristics* include the types of queries, update operations, transactions, and workflows that access the database (i.e., the *workload*

classes), their frequencies, their arrival rates during specific periods (e.g., main business hours vs. weekend vs. end-of-fiscal-year processing), their arrival patterns (e.g., typical sequences of different queries), their co-occurrence patterns (e.g., online transactions concurrently with certain batch jobs), the distribution of query parameter values, and so on.

- *Performance measures* include the *throughput* and *response times* (or properties of response-time distributions like quantiles) of different workload classes, but also metrics that capture the system's *dependability*, namely, *reliability* (e.g., probability of losing data by a permanent failure such as double or triple disk failures), *availability* (i.e., probability of being able to service requests at any timepoint in the presence of transient outages), *performability* (i.e., the performance level that can be sustained in degraded configurations, e.g., when servers or data replicas are temporarily unavailable), and ultimately, even capabilities to react to the system environment such as *resilience to security attacks*, graceful handling of denial-of-service attacks, countering attempts to breach data privacy, and so on.

If one had a reasonably accurate and complete model of the configuration-workload-performance relation, a self-managing system could, in principle, solve the following “inverse problem”:

Given specified goals for performance measures and the workload properties, find the lowest-cost configuration that satisfies the performance goals.

This would have to be solved dynamically whenever the workload exhibits major changes and necessitates adaptive reconfiguration. The goals in this setting should cover throughput, response time (e.g., a 95th quantile of at most 1 second), availability (e.g., expected downtime per year of at most 10 minutes), and other measures, leading to a more general notion of *service level agreements (SLA)* or *quality-of-service (QoS) guarantees*. The specified SLA/QoS requirements would drive the self-management procedures.

All three elements of the configuration-workload-performance function can refer to an entire system or to individual *components*. The latter is highly preferable: the configuration parameters are then restricted to the ones that are relevant for the specific component at hand, and the same modularization holds for workload properties and performance measures. At the system level, the configuration is the union of

the component-specific parameters, the workload at the system level is decomposed into workload properties for each component, and the performance observed at the component level is aggregated into system-level measures. As a consequence, predicting the system-wide performance requires *composability* of workload-characterization models and, most crucially, of performance-prediction models for the underlying components.

An inherent difficulty in achieving self-manageability is that many of the input parameters are not known *a priori*, may change rapidly thus posing difficulties to parameter estimation techniques, and are inherently difficult to model and thus bound to be incompletely captured. In particular, workload modeling is an extremely difficult task: queries, for example, can be modeled at different resolution levels from SQL statements down to internal operator trees (or actually, pipelined DAGs) or storage-level access operations; arrival rates can refer to steady state (long-term averages) or a specific look-ahead time horizon like the next hour or next minute; the dependencies in the arrival patterns of different query or transaction types can be determined by statistical correlations or by causal flow models via static program analysis of the application code.

Although the above discussion presents only a conceptual model, this framing does provide useful guidelines for approaching specific self-management issues. The salient aspects of the configuration-workload-performance framework and the general aim of solving for a suitable configuration suggest a methodology that is best characterized as an *observe-predict-react* cycle:

- *Observe*: Workload characteristics need to be observed as the system is running, so as to estimate parameters of the workload model or models. Observations may need to be collected at different resolution levels and time scales, and they must track drifts and anomalies in the workload evolution. This calls for a form of *introspection* or *self-monitoring*.
- *Predict*: As the current configuration is known and given the outcome of the observation step, one can aim to assess the performance in the near-term future, either with the current configuration as well as with various alternative configurations that could potentially improve performance. This calls for a notion of *what-if analysis*.
- *React*: Occasionally the system configuration needs to be reconsidered, in view of the observed workload

changes and projected near-term performance. In the case that explicit performance goals are given, the search for and assessment of new configurations could be triggered whenever one of the goals is violated (or about to be violated) with the current configuration. In the case without explicit goals, one or more objective functions need to be specified, so that whenever there is a significant loss in the objective function(s), the reaction step is invoked. The react step needs a smart search strategy to identify promising new configurations.

The observe-predict-react approach can be applied to *different time scales* based on the specific self-management task at hand from long-term capacity planning, which may be done relatively infrequently, down to real-time decisions, for example, about memory management, which requires all three steps at the resolution of minutes or even seconds to handle sudden memory-pressure situations. The three steps may be implemented as tools outside the database engine, or could be incorporated into the engine at *different integration levels*. Typically, short time scales mandate deeper integration, whereas long-term decisions could be made by external advisor tools.

Self-Management Paradigms

Ideally, the models and strategies for self-managing systems would follow a unified principle such as mathematical optimization theory, but such a “grand solution” is not in reach today. Instead, there are many diverse results on a variety of specific self-management issues; the state of the art is best characterized by a number of self-management paradigms: approaches that work well on paradigmatic example problems but bear the potential for being generalized into more broadly applicable principles. In the following, several such paradigms will be introduced, each with a general characterization and one or more exemplary use cases.

Trade-off Elimination: Tuning knobs exist because of trade-offs: there is no algorithm (or data structure) that performs near-optimal under all possible workloads, and therefore, systems are equipped with different options. However, sometimes it is possible to design an algorithm or a strategy that performs very well across the full spectrum of workloads, and this algorithm should be knob-free or only have second-order parameters which are uncritical or easy to set once across (almost) all workloads [19]. Such an

algorithm effectively eliminates the trade-off. Examples are modern cache-replacement algorithms (LRU-k, ARC, etc.) or B + -tree indexes for both exact-match lookups and range scans. In the case of quantitative tuning parameters such as disk block sizes or striping units, similar considerations may lead to robust techniques that can effectively eliminate knobs.

Static Optimization: Some self-tuning problems can be cast into mathematical optimizations with statically given input parameters. In principle, this opens up solutions based on combinatorial optimization methods such as branch-and-bound. Physical database design falls into this category; planning backup or replication procedures is another example. However, it is crucial to obtain input parameters and evaluate the objective function in a way that preserves the actual behavior of the database system. A major lesson from the research on physical design automation [8,9] is that a hand-crafted cost model, for assessing the quality of a particular design configuration, is bound to be inappropriate no matter how detailed and seemingly accurate it may be. The crux is that a separate cost model does not consider the actual behavior of the engine's query optimizer in selecting indexes for particular queries. Instead, the practically viable solutions references the query optimizer's cost model each time they need to assess the benefit of a design configuration for the given workload. To limit the computational overhead of these calls, thoroughly designed techniques for what-if assessment are needed (so as to estimate the benefit of a design configuration without actually building indexes). In addition, great care must be taken in the enumeration of candidate configurations. Although this is an offline optimization, the combinatorial explosion of the search space could easily lead to unacceptable run-time.

Stochastic Optimization: In some situations, albeit dealing with an offline optimization problem, the input parameters can only be characterized as random variables or by means of stochastic processes. System capacity planning falls into this category, for example, deciding how many disks are needed or how big a shared database cache should be in order to satisfy throughput and response time goals. For such problems, the established methodology is stochastic modeling, most importantly, queuing theory as the non-linear effects of resource contention under multi-user load are most critical [13,15]. For tractability, stochastic models often need to make simplifying assumptions,

and this entails the crucial issue of how accurate the model's predictions can still be. This concern can be addressed in two ways: (i) using more advanced mathematics to capture also non-standard situations or combining analytic models with simulations (e.g., to capture realistic inter-arrival time distributions rather than simply postulating an exponential distribution), and (ii) using stochastic models as a building block in a more comprehensive approach, where even somewhat inaccurate relative predictions are beneficial towards configuration decisions.

Online Optimization: Many self-managing problems obtain their inputs – workload parameters – only dynamically (but then accurately, not stochastically) and need to optimize configuration parameters as the workload evolves. These situations typically entail periodic or even continuous re-optimization, possibly in an incremental manner, and face tight timing constraints for finding the solution [2,3]. Memory governing for workspaces (e.g., for hash-joins or sorting) and database statistics management (e.g., for multidimensional histograms) fall into this category, the former having a time horizon of minutes or seconds, the latter with a typical reconsideration cycle of days but then requiring expensive database accesses. Inspired by online optimization theory, a possible approach to tackle these problems is to identify fast approximation techniques to obtain a viable solution. However, the details of the specific approach heavily depend on the problem at hand.

Feedback Control: When it is very difficult, if not impossible, to capture some aspect of the configuration-workload-performance relation in a causal model, the paradigm of feedback control loops offers a principled alternative even in the absence of causal understanding. Dynamic load control that adjusts the multiprogramming level (MPL, i.e., the maximum number of concurrent threads) of a server falls into this category. These methods consist of an admission control, to avoid overload effects that may result in performance thrashing, and a cancellation control to shed load when performance goals can no longer be met. This is crucially important especially for memory management, but potentially also for lock management and other resources. The main challenge that must be addressed here is that of sudden load surges: bursty arrivals of requests that require adjustment of MPL settings for smoother load (possibly even with workload-class-specific MPL limits). These transient effects are inherently difficult to model analytically, even with stochastic models. Feedback loops

treat the MPL settings as control variables that are adjusted based on simple differential equations over the measured performance values, the desired performance goals, and the control values. Control theory may be harnessed to ensure stability properties. Feedback control has been shown to work well for sufficiently simple functionality such as Web application servers [10].

Statistical Learning: Machine-learning models that use statistics for regression of continuous functions or classification with discrete labels are becoming increasingly attractive for self-tuning tasks [12,17]. Modern statistical learning methods can handle large numbers of input parameters (high-dimensional multivariate models) and can determine the most influential factors or can learn a predictor for an output variable. The only input is prior observations, sometimes along with manually assigned labels for training (if needed). In the context of self-managing database systems, the data are the system's event log: fine-grained information about configuration values (whenever they change), performance measurements, and also exceptions and potential problem situations. Statistical learning on this data can be used for diagnosis and root-cause analysis. A word of caution is in order, though: even if, in principle, all kinds of functions, labelings, and rankings can be learned, it is still an art to design the learning models and their feature spaces in the right way. So just like all the other self-management paradigms, statistical learning is not a panacea in itself.

Infrastructure

Self-management technology requires capabilities for adaptation, introspection, and self-healing. This in turn calls for a rich infrastructure in or around the database engine, so as to gather the necessary data and provide the mechanisms that will be enacted by the self-managing strategies. In the last few years, all modern database systems have addressed these requirements and provide rich facilities in this regard:

- Many internal algorithms (e.g., hash-joins or sorting) are *resource-adaptive*: they can continue running even if their resources, like workspace memory, are reduced at run-time. (But this alone does not provide self-management; in addition, strategies are needed for deciding when and how to reduce or increase resource assignments.)
- For *continuous self-monitoring*, light-weight techniques have been developed to identify and trace

relevant events (e.g., exceptions raised because of memory shortage) and aggregate large amounts of monitoring data. The difficulty that these techniques have successfully overcome is to do all this with very low run-time overhead so as not to adversely affect normal system operation [7].

- For self-healing, techniques have been added to isolate abnormal behavior and re-initialize potentially affected system components [4]. These approaches are eased by the fact that database systems generally follow the transactional paradigm for data accesses. By aborting ongoing transactions and sessions, resetting software components is greatly simplified. A similar argument holds for fail-over techniques among redundant servers (with data replication or shared disk storage), and this even works over large geographic distances to provide disaster recovery.

Future Directions

Self-management is an important topic across all kinds of computer systems. Ultimately, all IT systems should strive to become as easily usable as household appliances such as washing machines or TV sets. In some areas, the vision of administration-less and trouble-free solutions has almost been achieved, most notably, in storage systems [20]. However, database systems, with their very powerful languages like SQL and XQuery and their application-specific extensibility, have a much richer functionality than storage and thus face a much harder challenge.

In fact, database systems and their application workloads have become so complex that self-management is no longer just a desirable capability but will be a vital necessity in the long run. But despite the good progress on many issues of this theme, the quest for overriding principles has not yet achieved any breakthrough. This is partly due to the difficulty of the problems (and the quest is certainly ongoing); on the other hand, one could conjecture that the problem may, to some extent, be ill-defined given today's very sophisticated system architecture with their overwhelming richness of features and the limited set of abstractions.

This concern is reflected in considerations on componentizing database systems into building blocks with narrow interfaces and much fewer tuning choices [6,14], in building specialized data-management engines for particular application domains, or drastically limiting

the engine's options and functionality. Virtualization of resources is another trend towards simplifying system management, but its impact on database engine tuning is still unclear. A breakthrough may require radical departures from today's architectures as well as rethinking the functionality that is offered by the database system. Similar to the "design-for-recovery" position of [4], a completely new *design-for-manageability* approach may be needed.

Cross-references

- ▶ [Database Tuning using Combinatorial Search](#)
- ▶ [Database Tuning using Online Algorithms](#)
- ▶ [Database Tuning using Trade-off Elimination](#)

Recommended Reading

1. Ailamaki A (ed.) Special issue on self-managing database systems. *IEEE Data Eng. Bull.*, 29(3):2006.
2. Brown K.P., Mehta M., Carey M.J., and Livny M. Towards automated performance tuning for complex workloads. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994.
3. Bruno N. and Chaudhuri S. To tune or not to tune? A light-weight physical design alerter. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 499–510
4. Candea G., Brown A.B., Fox A., and Patterson D.A. Recovery-oriented computing: building multtier dependability. *IEEE Comput.*, 37(11):60–67, 2004.
5. Chaudhuri S., König A.C., and Narasayya V.R. SQLCM: a continuous monitoring framework for relational database engines. In Proc. 20th Int. Conf. on Data Engineering, 2004.
6. Chaudhuri S. and Narasayya V.R. An efficient cost-driven index selection tool for microsoft SQL server. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997.
7. Chaudhuri S. and Narasayya V. Self-tuning database systems: a decade of progress. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
8. Chaudhuri S., Narasayya V., and Syamala M. Bridging the application and DBMS profiling divide for database application developers. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
9. Chaudhuri S., and Weikum G. Rethinking database system architecture: towards a self-tuning RISC-style database system. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
10. Diao Y., Hellerstein J.L., Parekh S.S., Griffith R., Kaiser G.E., and Phung D.B. A control theory foundation for self-managing computing systems. *IEEE J. Select. Areas Commun.*, 23(12): 2213–2222, 2005.
11. Finkelstein S.J., Scholnick M., and Tiberio P. Physical database design for relational databases. *ACM Trans Database Syst.*, 13(1):91–128, 1988.
12. Jiang N., Villafane R., Hua K.A., Sawant A., and Prabhakara K. ADMiRe: an algebraic data mining approach to system performance analysis. *IEEE Trans. Knowl. Data Eng.*, 17(7):888–901, 2005.
13. Lazowska E.D., Zahorjan J., Scott Graham G., and Sevcik K.C. Quantitative system performance: computer analysis using

- queueing network models, Prentice-Hall, Englewood, Cliffs, NJ, 1984.
14. Lightstone S. Seven software engineering principles for autonomic computing development. *Innovations in Syst. and Softw. Eng.*, 3(1):71–74, 2007.
 15. Menase D.A. and Almeida V.A.F. Capacity Planning for Web Performance. Metrics, Models and Methods: Metrics, Models and Methods, Prentice-Hall, 2001.
 16. Reiner D.S. and Pinkerton T.B. A method for adaptive performance improvement of operating systems. In Proc. 18th ACM Symp. on Operating System Principles, 1981.
 17. Stillger M., Lohman G.M., Markl V., and Kandil M. LEO – DB2's LEarning Optimizer. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
 18. Weikum G., Hasse C., Moenkeberg A., and Zabback P. The COMFORT automatic tuning project. *Inform. Syst.*, 19(5): 381–432, 1994.
 19. Weikum G., Moenkeberg A., Hasse C., and Zabback P. Self-tuning database technology and information services: from Wishful thinking to viable engineering. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002.
 20. Wilkes J., Golding R.A., Staelin C., and Sullivan T. The HP AutoRAID hierarchical storage system. *ACM Trans. Comput. Syst.*, 14(1):108–136, 1996.

Self-Managing Database Systems

- ▶ [Self-Management Technology in Databases](#)

Self-Tuning Database Systems

- ▶ [Self-Management Technology in Databases](#)

Semantic Analysis of Video

- ▶ [Video Content Analysis](#)

Semantic Data Integration for Life Science Entities

ULF LESER

Humboldt University of Berlin, Berlin, Germany

Synonyms

[Object identification](#); [Data fusion](#); [Duplicate detection](#); [LSID](#)

Definition

An entity is the representation of a (not necessarily physical) real-world object, such as a gene, a protein, or a disease, within a database. To integrate information about the same entities from different databases, these representations must be analyzed to uncover the corresponding underlying objects. This process is called entity identification. A variation of entity identification is *duplicate detection*, which analyses two or more entities to determine whether they represent the same real-world object or not. Finally, *data fusion* is the process of generating a single, homogeneous representation from multiple, possibly inconsistent entities that represent the same real-world object.

When entities have globally unique *keys*, such as ISBN numbers in the case of books, entity identification and duplicate detection are simple. However, in life science databases, one usually has only descriptive information, such as the name or the sequence of a gene, which does not suffice to uniquely identify real-world objects. A *homonym* is a single name (or an ID) that identifies multiple, different objects. For instance, the term “ACE” may reference many different proteins, such as “angiotensin converting enzyme” or “acetylcholinesterase.” *Synonyms* are multiple names (or IDs) given to the same object.

Entity identification is particularly important in data curation, which is the (often manual) process of distilling a comprehensive description of a complex object from multiple data sources.

Historical Background

Duplicate detection has a long tradition in *census databases* where multiple representations of a person have to be identified to ensure reliable statistics. In the life sciences, the problem is particularly difficult, because many biological objects are much less stringently defined than, say, a human being. For instance, it is not clear whether two copies of a gene sequence on different sections of a chromosome should be considered the same gene or not, or if two highly similar and functionally identical genes in different species should be considered as different genes or not. At the other extreme, one often treats a gene and the protein it encodes as one object, especially in prokaryotes where differential splicing is almost non-existent, leading to a strict 1:1 relationship between genes and proteins. Thus, although a (DNA or protein) sequence is the fundamental identifying property of many biological

objects, it does not always lead to unique entity identification.

First calls for establishing world-wide standards to identify biological objects appeared in the early 1990s, when the Human Genome Project (HGP) quickly increased the amount of available information on genes and other molecular objects [4]. The HGP from the very start was an internationally distributed effort with no central organization that could have enforced consistent naming of objects or assignment of IDs to objects.

Standardization efforts were initiated in the domains of genes, proteins, and clones. Other areas where identification of objects is an important issue are small molecules, diseases, and species. One way to achieve standardized names was the installation of committees to define naming conventions for biological objects, such as the HUGO Gene Nomenclature Committee. On the other hand, some databases have become the de-facto standards for some types of biological objects, and their IDs are now commonly used as identifiers. An example is the use of UniProt-IDs as identifiers for proteins.

However, in many areas no standards exist or standards are not commonly used. Thus, many areas are facing the problem of identifying duplicates. The predominant approach is the comparison of biological sequences, i.e., DNA or protein sequences, using various variations of *edit distance* calculations. However, recent scientific discoveries, such as the importance of differential splicing (one gene forming several proteins) or the existence of paralogs (highly similar genes in the same genome with different function) render the pure usage of sequence similarity insufficient. Today, the method of choice for performing duplicate detection in a life sciences database therefore depends on the specific domain and the scientists’ and databases’ particular understanding of the biological object.

Foundations

Identity Versus Similarity

The entity identification problem exists for all types of objects in the life sciences. It is particularly pressing for genes and proteins. For most purposes one considers a gene to be a (not necessarily continuous) stretch of DNA, i.e., a sequence of the four nucleic acids, on a chromosome, which is – by some complex regulation mechanism – at times first transcribed into RNA and then translated into a protein. A protein is a molecule

consisting of a linear chain of amino acids forming a complex 3D structure. The translation procedure moves through the RNA and appends for each triple of nucleic acids one amino acid to the growing protein chain; thus, the translation of a given gene sequence into a protein is unique. The function of a protein mostly depends on the topology of its 3D structure and the properties of atoms exhibited at the surface of the structure [3].

The revelation of the function(s) of a protein is a crucial task in Bioinformatics and especially important for drug development. Because similar structure may hint toward similar function (independent of the species), a standard procedure for revealing the function of a human protein is to experimentally analyze the structure of a similar protein in another well-studied species, such as mouse or fruit-fly. Because the sequence of a gene determines the sequence of its protein(s), which in turn determines the structure of the protein, the same principle holds for genes: Two genes with similar sequence quite likely code for proteins with similar function.

With the function being the most important attribute of a gene/protein, the predominant question for semantic integration of genes and proteins is one of *similarity of sequences* rather than identity. The similarity of two biological sequences usually is measured by their (weighted) *edit distance* [9]. Two genes are considered as identical if their sequences are very similar, with the exact definition of “very similar” being a subject of much debate. The fundamental tool for entity identification and duplicate detection of genes and sequences therefore still is sequence comparison, using tools such as BLAST.

Naming Standards

Once a group of genes or proteins have been identified as “one real-world” object, one has to select a common name or ID for this group. For human genes, the Human Genome Organization (HUGO) installed the HUGO Gene Nomenclature Committee (HGNC), which is responsible for assigning names to human genes. Gene names are often complex, multi-term phrases that might include parts of the name of the person who discovered the gene, the phenotype it produces when a defect occurs, an inherited disease the gene is associated to, the chromosome and species where it first was detected, etc. However, the work of the HGNC is still widely ignored (especially in scientific

publications). For other species, similar bodies often were more successful in terms of standardization. The reason for their success is (i) the number of researchers working on a particular other species is usually much smaller than for humans, (ii) the number of genes in other species is much smaller (e.g., Yeast ~4,000 genes, human ~22,000 genes), and (iii) data collection was centralized early on; (see the Mouse Genome Database MGD for an example). Furthermore, certain database identifiers are commonly used to denote genes, especially those from the EntrezGene database.

No established naming convention exists for human proteins. The problem is worse than for genes, because on average every human gene codes for ~8 proteins by means of differential splicing. Very often database identifiers are used instead of spoken names. De-facto standards are IDs from the Protein Data Bank (PDB) and the Uniprot Knowledge Base (formerly known as SwissProt).

For DNA sequencing, chromosomes are broken into pieces, which are cultivated, copied, and distributed inside living bacteria, called clones. Sets of clones are gathered in libraries that often contain hundreds of thousands of clones. Physically, a library is a set of so-called plates with 10–100 dwells, each containing a particular clone. To avoid duplication of work, clones must be identified uniquely. However, after generation nothing is known about a particular clone except the dwell where it is hosted. Researchers therefore identify clones by the plate number and column/row number inside the library. Since re-distribution of clones into different, specialized libraries is commonplace, there are clones with more than 20 different IDs of this form (see the Genome Database (GDB) for examples).

To overcome the problem of object names, the *Object Management Group* (OMG) recently has defined the LSID – Life Science Identifiers – standard for “*persistent, location-independent, resource identifiers for uniquely naming biologically significant resources including species names, concepts, occurrences, genes or proteins, or data objects that encode information about them.*” An LSID identifier consists of a network identifier (usually the fixed term *urn:lsid*), an authority identifier (who defines this name), a namespace identifier, an object identifier, and a revision identifier (see discussion below on object versions). Whether or not this standard will be accepted by the community remains to be seen.

Evolution of Names

A particular problem with names and IDs is how to keep them stable and consistent. For instance, the question of which (of multiple) names of a particular gene is used in the literature is highly influenced by a kind of social-scientific “fashion” [10]. In biological databases, objects need to be merged and deleted, rendering existing IDs inconsistent. Furthermore, new findings may completely change what is known about a gene while keeping the ID unchanged, which makes previous studies based on the now outdated knowledge obsolete. This problem is usually solved by implementing a particular versioning model, which distinguishes major and versioned IDs; major ID always point to the most current version and a new version of the object is created with every update.

Key Applications

Semantic Integration of Entities

Duplicate detection is vital to ensure high data quality in integrated databases. It consists of two steps. First, multiple representations of same real-world entities have to be discovered. Second, for each group of duplicates, a uniform representation must be found [8]. Both steps are particularly difficult in the Life Sciences, because object definitions are vague (see above) and most biological data are obtained by complex experiments and are notoriously noisy [2].

Therefore, both tasks most often are performed manually, a process called *data curation*. It is common that large biological databases employ professional curators whose task it is to read new publications and to convert the most important information into some semi-structured database entry. There are also forms of community curation, where the correction of errors in databases through a web interface is possible for registered users [5].

The general problem with curation is that it is very costly and highly subjective. At the same time, deciding on the correct value given two diverging experimental results is usually impossible without further experiments. Consequently, a typical approach to data integration in this field is to omit the second step. The resulting architecture has been called “entity-based” or “multidimensional.” Different sources containing information about the same set of entities are considered as dimensions of the entities and are integrated into a schema similar to a *star schema* in *Data Warehouses*

[11]. The advantage and disadvantage is that the different views on entities are reported to the user in a logically separated manner; thus, she has the ability but also the obligation to select the appropriate values herself. Alternatively, mixtures of manual and automatic data fusion are used (see [1] for an example).

Entity Identification in Text

A related problem is the identification of object names in scientific publications, i.e., in English sentences. Named Entity Recognition (NER) is the problem of judging for a given set of terms within a document whether they form a gene name. A standard technique for solving NER is the usage of *classification* based on machine learning algorithms. Named Entity Normalization (NEN) is the problem of assigning a unique name to an entity in text once it has been identified as such. NEN typically is tackled using large *dictionaries* of names and some kind of fuzzy string matching method. See [6,7] for recent surveys on both tasks.

URL to Code

BLAST tool for search similar sequences in database:
<http://www.ncbi.nlm.nih.gov/BLAST/>

EntrezGene database: <http://www.ncbi.nlm.nih.gov/sites/entrez>

HUGO Gene Nomenclature Committee: <http://www.genenames.org/>

Mouse Genome Database: <http://www.informatics.jax.org/>

OMG Life Science Research Task Force: <http://www.omg.org/lsc/>

Protein Data Bank (PDB): <http://www.pdb.org/>

Uniprot knowledge base: <http://www.uniprot.org/>

Cross-references

- ▶ [Data Deduplication](#)
- ▶ [Information Integration](#)
- ▶ [Information Integration Techniques for Scientific Data](#)
- ▶ [Metadata Management and Resource Discovery](#)
- ▶ [Object Identity](#)
- ▶ [Scientific Databases](#)

Recommended Reading

1. Bhat T.N., Bourne P., Feng Z., Gilliland G., Jain S., Ravichandran V., Schneider B., Schneider K., Thanki N., and Weissig H, et al.

- The PDB data uniformity project. *Nucleic Acids Res.*, 29(1):214–218, 2001.
2. Brenner S.E. Errors in Genome Annotation. *Trends Genet.*, 15(4):132–133, 1999.
 3. Gibson G. and Muse S.V. A Primer of Genome Science. Sinauer Associates, Sunderland, MA, 2001.
 4. Karp P.D. Models of identifiers. In Proc. Second Meeting on Interconnection of Molecular Biology Databases. Cambridge, UK, 1995.
 5. Kingsbury D. Consensus, common entry, and community curation. *Nat. Biotechnol.*, 14:679, 1996.
 6. Krauthammer M. and Nenadic G. Term identification in the biomedical literature. *J. Biomed. Inform.*, 37(6):512–526, 2004.
 7. Leser U. and Hakenberg J. What Makes a Gene Name? Named Entity Recognition in the Biomedical Literature. *Briefings in Bioinformatics*, 6(4):357–369, 2005.
 8. Müller H., Naumann F., and Freytag J.-C. Data quality in genome databases. In Proc. Conf. on Information Quality, 2003.
 9. Smith T.F. and Waterman M.S. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.
 10. Tamames J. and Valencia A. The success (or not) of HUGO nomenclature. *Genome Biol.*, 7(5):402, 2006.
 11. Trissl S., Rother K., Müller H., Koch I., Steinke T., Preissner R., Frömmel C., and Leser U. Columba: an integrated database of proteins, structures, and annotations. *BMC Bioinformatics*, 6:81, 2005.

Semantic Data Model

DAVID W. EMBLEY

Brigham Young University, Provo, UT, USA

Synonyms

[Conceptual model](#); [Conceptual data model](#)

Definition

A *semantic data model* represents data in terms of named sets of objects, named sets of values, named sets of relationships, and constraints over these object, value, and relationship sets. The *semantics* of a semantic data model are the intensional declarations: the names for object, value, and relationship sets that indicate intended membership in the various sets and the declared constraints that the data should satisfy. The *data* of a semantic data model is extensional and consists of instances of object identifiers and values for object and value sets and of m -tuples of instances for m -ary relationship sets. The *model* of a semantic-data-model instance describes intensionally a real-world domain of interest. The modeling components of the

semantic data model specify the modeling elements from which a real-world model instances can be built.

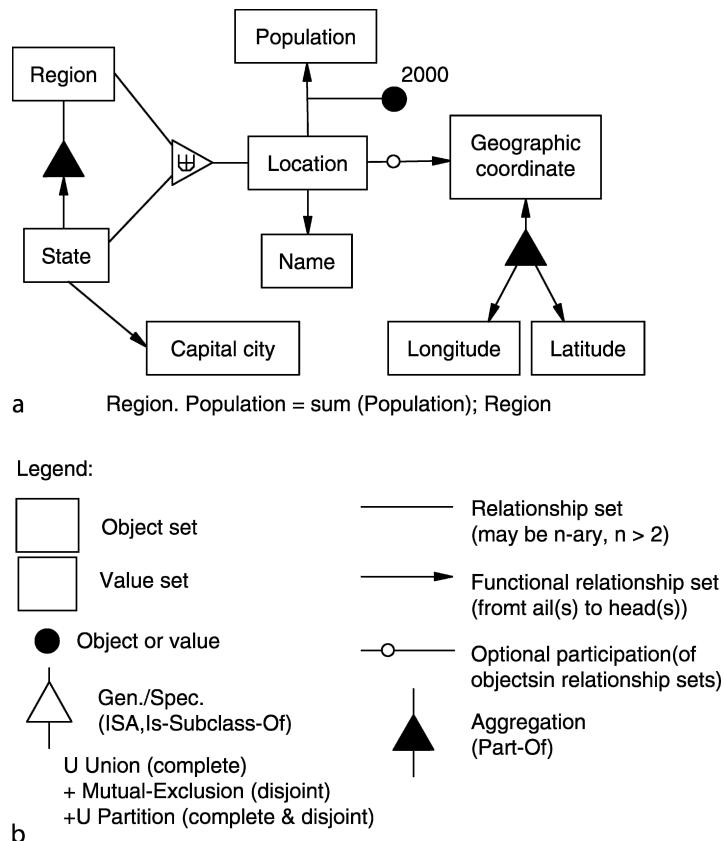
For a general description of semantic data models, see [1]. This article describes the generic properties of semantic data models and presents a representative collection of early semantic data models.

Key Points

[Figure 1a](#) gives a sample semantic data model. Semantic data models use graphical symbols to represent data semantics. Each semantic data model, however, has its own set of graphical symbols. The graphical symbols in [Fig. 1b](#) are meant to be generic-representative of the symbols used and illustrative of the kinds of data sets and constraints included in typical semantic data models.

The legend in [Fig. 1b](#) tells what each symbol means.

- A box with a solid border designates a set of objects. In [Fig. 1a](#) *State* designates the set of U.S. states (e.g., the state of California) and *Region* designates areas within the U.S. (e.g., the region of states in the Northeastern part of the U.S.).
- A box with a dashed border represents a set of values. In [Fig. 1a](#) *Capital City* designates the names of the capital cities of the U.S. states (e.g., “Sacramento” for California) and *Longitude* designates the set of longitude values for the geographic coordinates of locations (e.g., 120°4.9'W for the longitudinal part of the centerpoint of California).
- A large filled-in dot represents a single object or value. In [Fig. 1a](#) the object stands for the year 2000. Single objects or values can be thought of as singleton object or value sets – an object set or value set with one object.
- Relationship-set names can appear explicitly or can be a composition of the names of connected object and value sets. In [Fig. 1a](#) the names are all compositions (e.g., *State-CapitalCity* names the binary relationship set between *State* and *Capital City*). In general, relationship sets are m -ary ($n \geq 2$) (e.g., the *Location-2000-Population* relationship set is ternary).
- Constraints on relationship sets include functional/non-functional and mandatory/optional constraints. An arrowhead designates a functional relationship set from its tail(s) as domain space (s) to its head(s) as range space(s) (e.g., *Location-GeographicCoordinate* is functional from *Location*



Semantic Data Model. Figure 1. Sample semantic data model instance.

to *Geographic Coordinate*). A small “o” near a connection between an object (value) set and a relationship set allows for optional participation of the objects in the object set (values in the value set) in relationships in the relationship set (e.g., the “o” near *Location* in the *Location-GeographicCoordinate* relationship set makes the participation of location objects in the relationship set optional – a geographic coordinate for a location such as Northeast need not have a geographic coordinate).

- Generalization/specialization constraints, represented by a triangle, designate the specialization sets attached to the base of the triangle as subsets of the generalization set attached to the apex of the triangle (e.g., both the set of states and the set of regions are subset of the set of locations). If a union symbol (\cup) appears in the triangle, the generalization is a union of the specializations. If a mutual-exclusion symbol (+) appears, the specializations are pairwise non-intersecting. And if a partition symbol (\sqcup) appears, both a union and mutual-exclusion constraint hold, making the

constraint be a partition (e.g., in the semantic-data-model instance in Fig. 1, *Region* and *State* are mutually exclusive and the union constitutes *Location* – all the locations of interest for the semantic-data-model instance).

- Aggregation constraints, represented by a filled-in triangle, designate sub-part/super-part constraints. In Fig. 1a several states make up a region, and a longitude and latitude together constitute a geographic coordinate. The functional constraints associated with the aggregations allow a state to be part of at most one region and require that a longitude and latitude together correspond to one and only one geographic coordinate.

Formally, a populated semantic-data-model instance is an interpretation for a first-order language. Each object set and value set is a unary predicate (e.g., *State(x)* and *Latitude(x)*), and each *m*-ary relationship set is an *m*-ary predicate (e.g., *State-StateCapital(x, y)* and *Location-GeographicCoordinate(x, y)*). The domain for the interpretation is the set of object identifiers

and values in the populated semantic-data-model instance (or more generally the set of potential object identifiers and potential values for the semantic-data-model instance). The constraints are closed, well-formed formulas (e.g., $\forall x(State(x) \Rightarrow \exists!yState-StateCapital(x, y))$). A populated semantic-data-model instance whose data satisfies all the constraints is said to be a *model* – a valid semantic-data-model instance.

Cross-references

- ▶ Entity Relationship Model
- ▶ Extended Entity-Relationship Model
- ▶ Hierarchical Data Model
- ▶ Network Data Model
- ▶ Object Data Models
- ▶ Object-Role Modeling
- ▶ Ontology
- ▶ Unified Modeling Language

Recommended Reading

1. Peckham J. and Maryanski F. Semantic data models. ACM Comput. Surv., 20(3):153–189, September 1988.

elements $\langle ID_{ij}, n1_i, n2_j, R' \rangle$, with $n1_i \in G1$, $i = 1, \dots, N1$, $n2_j \in G2$, $j = 1, \dots, N2$ and R' the strongest *semantic relation* which is supposed to hold between the *concepts at nodes* $n1_i$ and $n2_j$.

A *mapping element* is a 4-tuple $\langle ID_{ij}, n1_i, n2_j, R \rangle$, $i = 1, \dots, N1; j = 1, \dots, N2$; where ID_{ij} is a unique identifier of the given mapping element; $n1_i$ is the i -th node of the first graph, $N1$ is the number of nodes in the first graph; $n2_j$ is the j -th node of the second graph, $N2$ is the number of nodes in the second graph; and R specifies a semantic relation which is supposed to hold between the concepts at nodes $n1_i$ and $n2_j$.

The *semantic relations* are within *equivalence* (=), *more general* (\supseteq), *less general* (\subseteq), *disjointness* (\perp) and *overlapping* (\sqcap). When none of the above mentioned relations can be explicitly computed, the special *idk* (I don't know) relation is returned. The relations are ordered according to decreasing binding strength, i.e., from the strongest (=) to the weakest (*idk*), with more general and less general relations having equal binding power. The semantics of the above relations are the obvious set-theoretic semantics.

Concept of a label is the logical formula which stands for the set of data instances or documents that one would classify under a label it encodes. *Concept at a node* is the logical formula which represents the set of data instances or documents which one would classify under a node, given that it has a certain label and that it is in a certain position in a graph.

Historical Background

An ontology typically provides a vocabulary that describes a domain of interest and a specification of the meaning of terms used in the vocabulary. Depending on the precision of this specification, the notion of ontology encompasses several data and conceptual models, for example, classifications, database schemas, or fully axiomatized theories. In open or evolving systems, such as the semantic web, different parties would, in general, adopt different ontologies. Thus, just using ontologies, just like using XML, does not reduce heterogeneity: it raises heterogeneity problems to a higher level. Ontology matching is a plausible solution to the semantic heterogeneity problem faced by information management systems. Ontology matching aims at finding correspondences or mapping elements between semantically related entities of the input ontologies. These mapping elements can be used for various tasks, such as ontology merging, query answering, data

Semantic Image Retrieval

- ▶ Annotation-based Image Retrieval

Semantic Inference in Audio

- ▶ Audio Content Analysis

Semantic Mapping Composition

- ▶ Schema Mapping Composition

Semantic Matching

FAUSTO GIUNCHIGLIA, PAVEL SHVAIKO,
MIKALAI YATSKEVICH
University of Trento, Trento, Italy

Definition

Semantic matching: given two graph representations of ontologies $G1$ and $G2$, compute $N1 \times N2$ *mapping*

translation, etc. Thus, matching ontologies enables the knowledge and data expressed in the matched ontologies to interoperate [6].

Many diverse solutions of matching have been proposed so far, see [7,17,18] for recent surveys which addressed the matching problem from different perspectives, including databases, artificial intelligence and information systems; while the major contributions of the last decades are provided in [2,14,19]. Some examples of individual approaches addressing the matching problem can be found in [4,5,8,15,16]. (See <http://www.ontologymatching.org> for a complete information on the topic.) Finally, ontology matching has been given a book account in [6]. This work provided a uniform view on the topic with the help of several classifications of the available methods, discussed these methods in detail, etc. In particular, the matching methods are primarily classified and further detailed according to (i) the input of the algorithms, (ii) the characteristics of the matching process and (iii) the output of the algorithms.

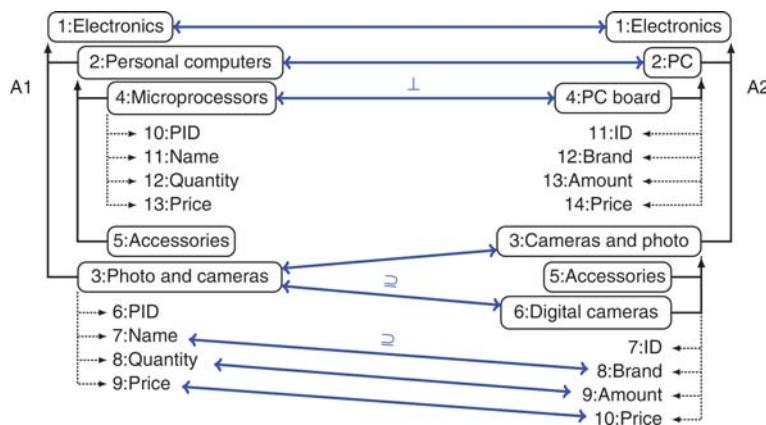
The work in [10] mixed the process dimension of matching together with the output dimension and classified matching approaches into *syntactic* and *semantic*. Syntactic are those approaches that rely on purely syntactic matching methods, e.g., edit distance between strings, tree edit distance. The semantic category, in turn, represents methods that work with concepts and compare their meanings in order to compute mapping elements. However, these have been also constrained

by a second condition dealing with the output dimension: syntactic techniques return coefficients in the [0 1] range, while semantic techniques return logical relations, such as equivalence, subsumption (and justified by deductive techniques for instance). The work in [4] provided a first implementation of semantic matching.

Foundations

In order to motivate the matching problem two simple XML schemas are used. These are represented as trees in Fig. 1 and exemplify one of the possible situations which arise, for example, when resolving a schema integration task. Suppose an e-commerce company A1 needs to finalize a corporate acquisition of another company A2. To complete the acquisition, databases of the two companies have to be integrated. The documents of both companies are stored according to XML schemas A1 and A2, respectively. A first step in integrating the schemas is to identify candidates to be merged or to have taxonomic relationships under an integrated schema. This step refers to a process of ontology (schema) matching. For example, the elements with labels *Personal_Computers* in A1 and *PC* in A2 are the candidates to be merged, while the element with label *Digital_Cameras* in A2 should be subsumed by the element with label *Photo_and_Cameras* in A1.

Consider semantic matching as first motivated in [10] and implemented within the S-Match system [13]. Specifically, a schema-based solution is discussed,



Semantic Matching. **Figure 1.** Two simple XML schemas. The XML elements are shown in rectangles with rounded corners, while attributes are shown without them. Numbers before the labels of tree nodes are the unique identifiers of the XML elements and attributes. In turn, the mapping elements are expressed by arrows. By default, their relation is $=$; otherwise, these are mentioned above the arrows.

where only the schema information is exploited. It is assumed that all the data and conceptual models, e.g., classifications, database schemas, ontologies, can be generally represented as graphs. This allows for the statement and solution of a *generic (semantic) matching problem* independently of specific conceptual or data models, very much along the lines of what is done, for example, in Cupid [15].

The semantic matching takes as input two graph representations of ontologies and returns as output logical relations, e.g., equivalence, subsumption (instead of computing coefficients rating match quality in the [0 1] range, as it is the case with other approaches, e.g., [15,16]), which are supposed to hold between the nodes in the graphs. The relations are determined by (i) expressing the entities of the ontologies as logical formulas, and (ii) reducing the matching problem to a logical validity problem. In particular, the entities are translated into logical formulas which explicitly express the concept descriptions as encoded in the ontology structure and in external resources, such as WordNet. (<http://wordnet.princeton.edu/>) This allows for a translation of the matching problem into a logical validity problem, which can then be efficiently resolved using (sound and complete) state of the art satisfiability solvers.

Consider tree-like structures, e.g., classifications, and XML schemas. Real-world ontologies are seldom trees, however, there are (optimized) techniques, transforming a graph representation of an ontology into a tree representation, e.g., the graph-to-tree operator of Protoplasm [3]. From now on it is assumed that a graph-to-tree transformation can be done by using existing systems, and therefore, the focus is on other issues instead.

Consider Fig. 1. “C” is used to denote concepts of labels and concepts at nodes. Also “C₁” and “C₂” are used to distinguish between concepts of labels and concepts at nodes in tree 1 and tree 2, respectively. Thus, in A1, C₁_{Photo_and_Cameras} and C₁₃ are, respectively, the concept of the label *Photo_and_Cameras* and the concept at node 3. Finally, in order to simplify the presentation whenever it is clear from the context, it is assumed that the formula encoding the concept of label is the label itself. Thus, for example in A2, *Cameras_and_Photo*₂ is a notational equivalent of C₂_{Cameras_and_Photo}.

The algorithm inputs two ontologies and outputs a set of mapping elements in four macro steps. The first

two steps represent the pre-processing phase. The third and the fourth steps are the element level and structure level matching, respectively. (Element level matching techniques compute mapping elements by analyzing entities in isolation, ignoring their relations with other entities. Structure level techniques compute mapping elements by analyzing how entities are related together.)

Step 1. For all labels L in the two trees, compute concepts of labels. The labels at nodes are viewed as concise descriptions of the data that is stored under the nodes. The meaning of a label at a node is computed by taking as input a *label*, analyzing its real-world semantics, and returning as output a *concept of the label*, C_L. Thus, for example, C_{Cameras_and_Photo} indicates a shift from the natural language ambiguous label *Cameras_and_Photo* to the concept C_{Cameras_and_Photo} which codifies explicitly its intended meaning, namely the data which is about cameras and photo. Technically, concepts of labels are codified as propositional logical formulas [9]. First, labels are chunked into *tokens*, e.g., *Photo_and_Cameras* → ⟨photo, and, cameras⟩; and then, *lemmas* are extracted from the tokens, e.g., *cameras* → *camera*. Atomic formulas are WordNet *senses* of lemmas obtained from single words (e.g., cameras) or multiwords (e.g., digital cameras). Complex formulas are built by combining atomic formulas using the connectives of set theory. For example, C₂_{Cameras_and_Photo} = ⟨Cameras, senses_{WN#2}⟩ ∪ ⟨Photo, senses_{WN#1}⟩, where senses_{WN#2} is taken to be disjunction of the two senses that WordNet attaches to *Cameras*, and similarly for *Photo*. The natural language conjunction “and” has been translated into the logical disjunction “ \sqcup ”

Step 2. For all nodes N in the two trees, compute concepts at nodes. During this step the meaning of the positions that the labels at nodes have in a tree is analyzed. By doing this, concepts of labels are *extended* to *concepts at nodes*, C_N. This is required to capture the knowledge residing in the structure of a tree, namely the context in which the given concept at label occurs. For example, in A2, by writing C₆ it is meant the concept describing all the data instances of the electronic photography products which are digital cameras. Technically, concepts of nodes are written in the same propositional logical language as concepts of labels. XML schemas are hierarchical structures where the path from the root to a node uniquely identifies that node (and also its meaning). Thus, following an

access criterion semantics, the logical formula for a concept at node is defined as a conjunction of concepts of labels located in the path from the given node to the root. For example, $C2_6 = Electronics_2 \sqcap Cameras_and_Photo_2 \sqcap Digital_Cameras_2$.

Step 3. For all pairs of labels in the two trees, compute relations among atomic concepts of labels. Relations between concepts of labels are computed with the help of a library of element level semantic matchers. These matchers take as input two atomic concepts of labels and produce as output a semantic relation between them. Some of them are re-implementations of the well-known matchers used, e.g., in Cupid. The most important difference is that these matchers return a semantic relation (e.g., $=$, \sqsupseteq , \sqsubseteq), rather than an affinity level in the $[0, 1]$ range, although sometimes using customizable thresholds.

The element level semantic matchers are briefly summarized in Table 1. The first column contains the names of the matchers. The second column lists the order in which they are executed. The third column introduces the matcher's approximation level. The relations produced by a matcher with the first approximation level are always correct. For example, *name* \sqsupseteq *brand* returned by the WordNet matcher. In fact, according to WordNet *name* is a hypernym (superordinate word) of *brand*. In WordNet *name* has 15 senses and *brand* has 9 senses. Some sense filtering techniques are used to discard the irrelevant senses for the given context, see [13] for details. Notice that matchers are executed following the order of increasing approximation. The fourth column reports the matcher's type, while the fifth column describes the matcher's input. As from Table 1, there are two main categories of matchers. *String-based* matchers have two labels as input. These compute only equivalence relations (e.g., equivalence holds if the weighted distance between the input strings is lower than a threshold). *Sense-based*

matchers have two WordNet senses as input. The WordNet matcher computes equivalence, more/less general, and disjointness relations. The result of step 3 is a matrix of the relations holding between atomic concepts of labels. A part of this matrix for the example of Fig. 1 is shown in Table 2.

Step 4. For all pairs of nodes in the two trees, compute relations among concepts at nodes. During this step, initially the tree matching problem is reformulated into a set of node matching problems (one problem for each pair of nodes). Then, each node matching problem is translated into a propositional validity problem. Semantic relations are translated into propositional connectives in an obvious way, namely: equivalence ($=$) into equivalence (\leftrightarrow), more general (\sqsupseteq) and less general (\sqsubseteq) into implication (\leftarrow and \rightarrow , respectively) and disjointness (\perp) into negation (\neg) of the conjunction (\wedge). The criterion for determining whether a relation holds between concepts at nodes is the fact that it is entailed by the premises. Thus, it is necessary to prove that the following formula:

$$\text{axioms} \rightarrow \text{rel}(\text{context}_1, \text{context}_2) \quad (1)$$

is valid, namely that it is *true* for all the truth assignments of all the propositional variables occurring in it. context_1 is the concept at node under consideration in tree 1, while context_2 is the concept at node under consideration in tree 2. rel (within $=$, \sqsubseteq , \sqsupseteq , \perp) is the semantic relation (suitably translated into a propositional connective) to be proved to hold between context_1 and context_2 . The *axioms* part is the conjunction of all the relations (suitably translated) between atomic concepts of labels mentioned in context_1 and context_2 . The validity of formula (1) is checked by proving that its negation is unsatisfiable. Specifically, it is done, depending on a matching task, either by using ad hoc reasoning techniques or standard propositional satisfiability solvers.

Semantic Matching. Table 1. Element level semantic matchers

Matcher name	Execution order	Approximation level	Matcher type	Schema info
WordNet	1	1	Sense-based	WordNet senses
Prefix	2	2	String-based	Labels
Suffix	3	2	String-based	Labels
Edit distance	4	2	String-based	Labels
Ngram	5	2	String-based	Labels

From the example in Fig. 1, trying to prove that $C2_6$ is less general than $C1_3$, requires constructing formula (2), which turns out to be unsatisfiable, and therefore, the less general relation holds.

$$\begin{aligned} & ((Electronics_1 \leftrightarrow Electronics_2) \wedge (Photo_1 \leftrightarrow Photo_2) \wedge \\ & (Cameras_1 \leftrightarrow Cameras_2) \wedge \\ & (Digital_Cameras_2 \rightarrow Cameras_1)) \wedge (Electronics_2 \wedge \\ & (Cameras_2 \vee Photo_2) \wedge Digital_Cameras_2) \wedge \\ & \neg(Electronics_1 \wedge (Photo_1 \vee Cameras_1)) \end{aligned} \quad (2)$$

A part of this matrix for the example of Fig. 1 is shown in Table 3.

Finally, notice that the algorithm returns $N1 \times N2$ correspondences, therefore the cardinality of mapping elements is *one-to-many*. Also, these, if necessary, can be decomposed straightforwardly into mapping elements with the *one-to-one* cardinality.

Key Applications

Semantic matching is an important operation in traditional metadata intensive applications, such as *ontology integration*, *schema integration*, or *data warehouses*. Typically, these applications are characterized by heterogeneous structural models that are analyzed and matched either manually or semi-automatically at design time. In such applications matching is a prerequisite of running the actual system. A line of applications that can be characterized by their dynamics, e.g., *agent communication*, *peer-to-peer information sharing*, *web service composition*, is emerging. Such applications, contrary to traditional ones, require (ultimately) a run

Semantic Matching. Table 2. The matrix of semantic relations holding between atomic concepts of labels

	Cameras ₂	Photo ₂	Digital_Cameras ₂
Photo ₁	idk	=	idk
Cameras ₁	=	idk	⊓

Semantic Matching. Table 3. The matrix of semantic relations holding between concepts at nodes (the matching result)

	C2 ₁	C2 ₂	C2 ₃	C2 ₄	C2 ₅	C2 ₆
C1 ₃	⊒	idk	=	idk	⊓	⊓

time matching operation and take advantage of more explicit conceptual models [18].

Future Directions

Future work includes development of a fully-fledged *iterative* and *interactive* semantic matching system. It will improve the quality of the mapping elements by iterating and by focusing user's attention on the critical points where his/her input is maximally useful. Initial steps have already been done in this direction by discovering automatically *missing background knowledge* in ontology matching tasks [11]. Also, an *evaluation methodology* is needed, capable of estimating quality of the mapping elements between ontologies with hundreds and thousands of nodes. Initial steps have already been done as well; see [1,12] for details. Here, the key issue is that in these cases, specifying reference mapping elements manually is neither desirable nor feasible task, thus a semi-automatic approach is needed.

Experimental Results

In general, for the semantic matching approach, there is an accompanying experimental evaluation in the corresponding references. Also, there is the Ontology Alignment Evaluation Initiative (OAEI), (<http://oaei.ontologymatching.org/>) which is a coordinated international initiative that organizes the evaluation of the increasing number of ontology matching systems. The main goal of the Ontology Alignment Evaluation Initiative is to be able to compare systems and algorithms on the same basis and to allow anyone for drawing conclusions about the best matching strategies. From such evaluations, matching system developers can learn and improve their systems.

Data Sets

A large collection of datasets commonly used for experiments can be found at: <http://oaei.ontologymatching.org/>.

URL to Code

The OntologyMatching.org contains links to a number of ontology matching projects which provide code for their implementations of the matching operation: <http://www.ontologymatching.org/>.

Cross-references

- ▶ [Data Integration](#)
- ▶ [Data models \(including semantic data models\)](#)
- ▶ [Mobile and ubiquitous data management](#)

Recommended Reading

1. Avesani P., Giunchiglia F., and Yatskevich M. A large scale taxonomy mapping evaluation. In Proc. Fourth Int. Semantic Web Conf., 2005, pp. 67–81.
2. Batini C., Lenzerini M., and Navathe S. A comparative analysis of methodologies for database schema integration. ACM Comput. Surv., 18(4):323–364, 1986.
3. Bernstein P., Melnik S., Petropoulos M., and Quix C. Industrial-strength schema matching. ACM SIGMOD Rec., 33(4):38–43, 2004.
4. Bouquet P., Serafini L., and Zanobini S. Semantic coordination: a new approach and an application. In Proc. Second Int. Semantic Web Conf., 2003, pp. 130–145.
5. Doan A., Madhavan J., Dhamankar R., Domingos P., and Halevy A.Y. Learning to match ontologies on the Semantic Web. VLDB J., 12(4):303–319, 2003.
6. Euzenat J. and Shvaiko P. Ontology Matching. Springer, 2007.
7. Gal A. Why is schema matching tough and what can we do about it? ACM SIGMOD Rec., 35(4):2–5, 2006.
8. Gal A., Anaby-Tavor A., Trombetta A., and Montesi D. A framework for modeling and evaluating automatic semantic reconciliation. VLDB J., 14(1):50–67, 2005.
9. Giunchiglia F., Marchese M., and Zaihrayeu I. Encoding classifications into lightweight ontologies. J. Data Semantics, 8:57–81, 2007.
10. Giunchiglia F. and Shvaiko P. Semantic Matching. Knowl. Eng. Rev., 18(3):265–280, 2003.
11. Giunchiglia F., Shvaiko P., and Yatskevich M. Discovering missing background knowledge in ontology matching. In Proc. 17th European Conf. on Artificial Intelligence, 2006, pp. 382–386.
12. Giunchiglia F., Yatskevich M., Avesani P., and Shvaiko P. A large scale dataset for the evaluation of ontology matching systems. Knowl. Eng. Rev., 23:1–22, 2008.
13. Giunchiglia F., Yatskevich M., and Shvaiko P. Semantic matching: algorithms and implementation. J. Data Semantics, 9:1–38, 2007.
14. Larson J., Navathe S., and Elmasri R. A theory of attributed equivalence in databases with application to schema integration. IEEE Trans. Software Eng., 15(4):449–463, 1989.
15. Madhavan J., Bernstein P., and Rahm E. Generic schema matching with Cupid. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 48–58.
16. Noy N. and Musen M. The PROMPT suite: interactive tools for ontology merging and mapping. Int. J. Hum. Comput. Stud., 59 (6):983–1024, 2003.
17. Rahm E. and Bernstein P. A survey of approaches to automatic schema matching. VLDB J., 10(4):334–350, 2001.
18. Shvaiko P. and Euzenat J. A survey of schema-based matching approaches. J. Data Semantics, 4:146–171, 2005.
19. Spaccapietra S. and Parent C. Conflicts and correspondence assertions in interoperable databases. ACM SIGMOD Rec., 20(4):49–54, 1991.

Semantic Modeling and Knowledge Representation for Multimedia Data

EDWARD Y. CHANG

Google Research, Mountain View, CA, USA

Synonyms

[Multimedia information retrieval](#); [Image/Video/Music search](#)

Definition

Semantic modeling and knowledge representation is essential to a multimedia information retrieval system for supporting effective data organization and search. Semantic modeling and knowledge representation for multimedia data (e.g., imagery, video, and music) consists of three steps: *feature extraction*, *semantic labeling*, and *features-to-semantics mapping*. Feature extraction obtains perceptual characteristics such as color, shape, texture, salient-object, and motion features from multimedia data; semantic labeling associates multimedia data with cognitive concepts; and features-to-semantics mapping constructs correspondence between perceptual features and cognitive concepts. Analogically to data representation for text documents, improving semantic modeling and knowledge representation for multimedia data leads to enhanced data organization and query performance.

Historical Background

The principal design goal of a multimedia information retrieval system is to return data (images, video clips, or music) that accurately match users' queries (for example, a search for pictures of a deer). To achieve this design goal, the system must first comprehend a user's query concept thoroughly, and then find data in the low-level input space (formed by a set of perceptual features) that match the concept accurately. For traditional relational databases, a query concept is explicitly specified by a user using SQL. For multimedia information retrieval, however, articulating a query concept (e.g., a deer) using low level features (e.g., color, shape, texture, and salient-object features) is infeasible.

Semantic modeling and knowledge representation thus plays a key role in query-concept formulation and query processing for a multimedia query.

The QBIC system [8] introduced in 1995 is the first query-by-example system. QBIC uses color histograms to represent an image/video clip; two images/clips containing similar color histograms are considered to be similar. Such knowledge representation for multimedia data is clearly inadequate. In the subsequent 5 years, many researchers in the *signal processing* and *computer vision* communities proposed techniques to extract perceptual features, such as textures, shapes, and segments of objects, for improve image representation (see [13] for a survey). At the same time, the query-by-example paradigm was applied also to music retrieval.

Query by *just one example* was soon discovered insufficient to represent a query concept. Relevance feedback, a query refinement technique developed by the information retrieval community in the 1970's [12], was then borrowed to provide additional examples to augment the shortcoming of knowledge under-representation. In 2001, the work of [14] showed that relevance feedback could be much improved by using the *kernel methods* [1] with *active learning*. The kernel methods project data from their input space formed by perceptual features to a much higher (possibly infinite) dimensional space, where a linear classifier can be learned to separate desired data (with respect to the query) from the others. The kernel methods enjoy both rich semantic modeling (the linear class boundary in a high-dimensional space represents a non-linear boundary in the input space) and computational efficiency (computation is performed in the projected, linear space). Active learning is applied to select the most ambiguous and diversified training instances along the class boundary to query the user for labels. Once these training instances have been labeled, maximal information is gained for refining the class boundary. This process of active learning continues until the search result is satisfactory. In order to further improve the effectiveness of query-concept learning through active learning, keywords (tagged by users [9] or obtained from query logs [10]) were subsequently integrated into the semantic modeling and knowledge representation framework.

Over a decade of research since QBIC, though productive, has not yielded a large-scale real-world

deployment of multimedia information retrieval system. The key reason is that semantic modeling and knowledge representation for multimedia data is intrinsically inter-disciplinary. Its success demands collaborative effort from researchers of *signal processing*, *computer vision*, *machine learning*, and *databases*. Recent works in addressing issues of perceptual similarity [11] and scalability in statistical learning [5] are interdisciplinary approaches that hold promises to lead to a Web-scale deployment. The survey conducted by [6] provides a complementary view on the historical background.

Foundations

Semantic Modeling

There are two realistic ways for users to specify a multimedia query semantic: *query by keywords* and *query by examples*. In order to support query by keywords, *semantic annotation* provides data with semantic labels (for example, landscape, sunset, animals, and so forth). Several researchers (e.g., [2]) have proposed semi-automatic annotation methods to propagate keywords from a small set of annotated images to the other images. Although semantic annotation can provide some relevant query results, annotation is often subjective and narrowly construed. When it is, query performance may be compromised. To thoroughly understand a query concept, with all of its semantics and subjectivity, a system must obtain the target concept from the user directly via *query-concept learning*. Semantic annotation can assist, but not replace, query-concept learning.

Both semantic annotation and query-concept learning require mapping features to semantics. This semantic modeling consists of three steps. First, a set of perceptual features (e.g., color, texture and shape) is extracted from each training instance. Second, each training feature-vector \mathbf{x}_i is assigned semantic labels \mathbf{g}_i . Third, a classifier $f(\cdot)$ is trained by a supervised learning (or semi-supervised learning) algorithm, based on the labeled instances, to predict the class labels of a query instance \mathbf{x}_q . Given a query instance \mathbf{x}_q represented by its low-level features, the semantic labels \mathbf{g}_q of \mathbf{x}_q can be predicted by $\mathbf{g}_q = f(\mathbf{x}_q)$. (About how multimedia data and knowledge can be represented is discussed in the Knowledge Representation section.)

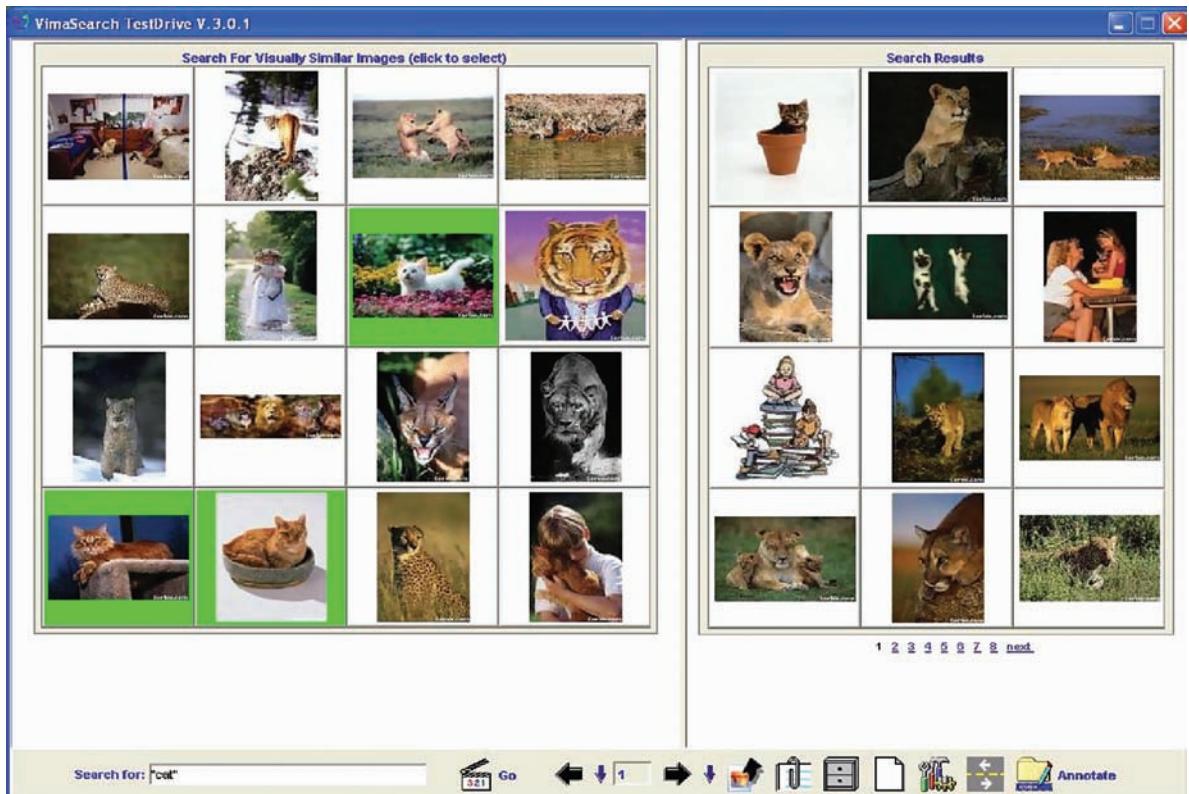
At first it might seem that traditional supervised learning methods could be directly applied to perform *semantic annotation* and *query-concept learning*. Unfortunately, traditional learning algorithms are not adequate to deal with the technical challenges posed by these two tasks. To illustrate, let D denote the number of low-level features. Let N denote the number of training instances, N^+ the number of positive training instances, and N^- the number of negative training instances ($N = N^+ + N^-$). And let U denote the number of unlabeled instances in the repository. Three major technical challenges arise:

1. *Scarcity of training data.* The features-to-semantics mapping problem often comes up against the $D > N$ challenge. For instance, in the query-concept learning scenario, the number of low-level features that characterize an image (D) is greater than the number of training instances that a user can provide (N) via her query history or relevance feedback. The theories underlying “classical” data analysis are based on the assumptions that $D < N$, and N approaches infinity. But when $D > N$, the

basic methodology which was used in the classical situation is not similarly applicable [7].

2. *Imbalance of training classes.* The target class in the training pool is typically outnumbered by the non-target classes ($N^- >> N^+$). When the prior of the non-target class dominates the target class, a class prediction favors the non-target class. This skew can substantially reduce recall in search performance [16].
3. *Scalability.* A typical value of D can be in the order of hundreds, and U can be millions or even billions. Scalability challenges arise in at least two areas. First, searching data among U instances in a high-dimensional space is inefficient [3]. Second, when $U >> N$, training data may under-represent the knowledge required to model semantics.

Effective techniques for addressing the above challenges are inter-disciplinary. The *signal processing* and *computer vision* communities devise algorithms to extract useful features to represent multimedia data. The *machine learning* community develops models that can map features to semantics both effectively and



Semantic Modeling and Knowledge Representation for Multimedia Data. Figure 1. Cat query initial screen.

efficiently. The *database* community improves indexing, metadata fusion, and query processing techniques to deal with scalability issues. All these endeavors may consult experts in *neural processing* or *cognitive science* (e.g., [15]) to develop representations and models that fit human perception.

Knowledge Representation

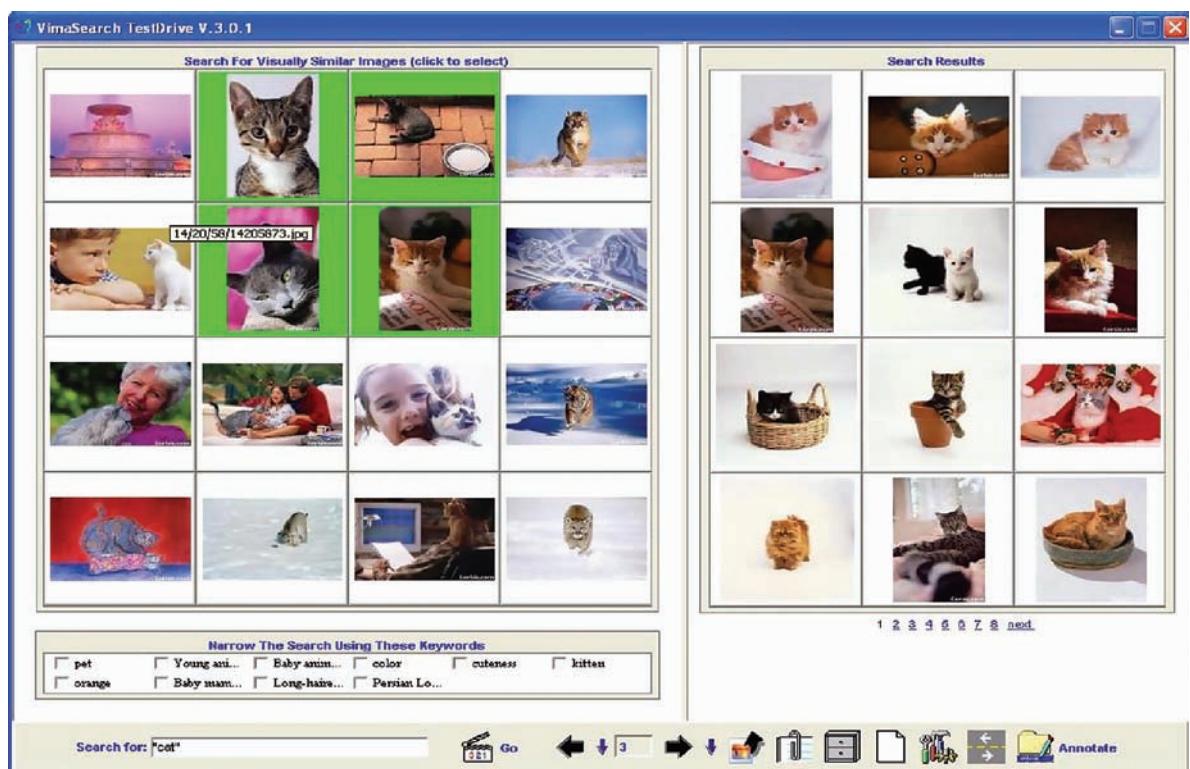
As mentioned, a piece of multimedia data can be represented at two levels: low-level features and high-level semantics/concepts. A set of low-level features consists of perceptual features, and these features can be put in the form of a vector or a bag. High-level concepts are organized into an ontology structure, depicting relationship between concepts. In between, descriptors can be formulated either explicitly or implicitly to provide building blocks for low-level to high-level mapping and reasoning. For instance, a high-level ski concept can be formed by descriptors of snow, ski equipment, and people. Each of these descriptors is in turn composed of color, texture, shape, or salient-point features. Texts when available can be used to augment low-level perceptual features (e.g., using word “white” to depict the color of the mountain), to

label descriptors (e.g., snow), or to directly annotate high-level semantics/concepts (e.g., ski). Statistical methods such as SVMs and Latent Semantic Analysis techniques (e.g., LDA [4]) can be employed to perform mapping between the three levels.

Efforts of standardizing knowledge representation have been embarked on for over a decade by academia and industry. For instance, digital cameras save JPEG files with EXIF (Exchangeable Image File) data. EXIF records camera settings, scene information, and time (and location where a photo is taken in the near future). DOLCE devises descriptive ontology for linguistic and cognitive engineering. MPEG-7 proposes different description granularity to depict multimedia data. Standard knowledge representation is essential for supporting metadata exchange and system interoperability.

Key Applications

The launches of photo and video sharing sites such as Flickr, Google Photos, and YouTube between 2002 and 2008 renewed the interest on multimedia data management. The following applications are in high demand to manage large-scale multimedia data repositories:



Semantic Modeling and Knowledge Representation for Multimedia Data. Figure 2. Cat query after iteration #2.

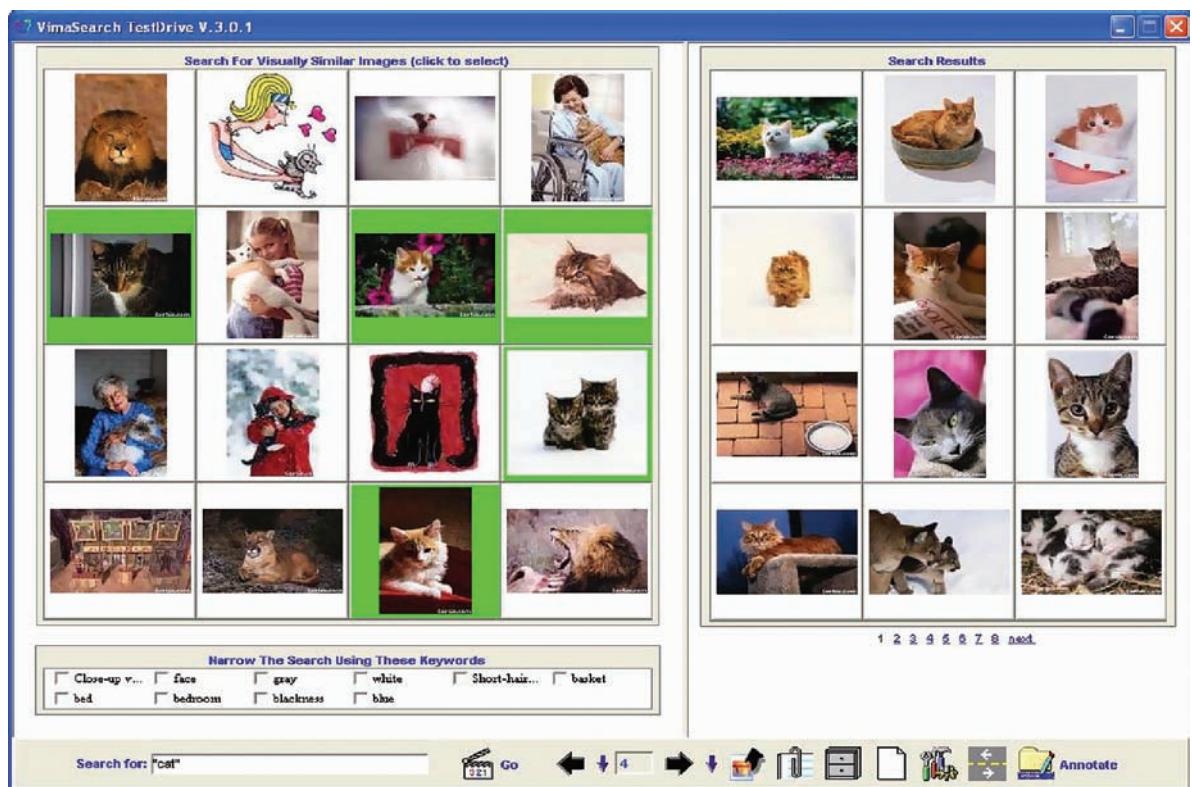
1. Content-based Video, Image, Music Search Engines
2. Copy Right Infringement Detection
3. Multimedia Digital Libraries
4. Semi-automatic Photo/Video Annotation/Classification

An application scenario is used to illustrate how aforementioned science fundamentals can improve multimedia information retrieval. Figures 1–3 show an example query using a *Perception-based Image Retrieval* (PBIR) prototype developed at UC Santa Barbara. The figures demonstrate how a query concept is learned in an iterative process by the PBIR search engine to improve search results. The user interface shows two frames. The frame on the left-hand side is the feedback frame, on which the user marks images relevant to his or her query concept. On the right-hand side, the search engine returns what it interprets as matching this far from the image database.

Most images were annotated by users. To query “cat,” one first enters the keyword *cat* in the query box to get the first screen of results in Fig. 1. The right-side frame shows a couple of images containing

domestic cats, but several images containing tigers or lions. This is because many tiger/lion images were annotated with “wild cat” or “cat.” To disambiguate the concept, the user clicks on a couple of domestic cat images on the feedback frame (left side, in gray/green borders). The search engine refines the class boundary accordingly, and then returns the second screen in Fig. 2. In this figure, the images in the result frame (right side) have been much improved. All returned images contain a domestic cat or two. After performing another couple of rounds of feedback to make some further refinements, more satisfactory results are shown in Fig. 3.

This example illustrates three critical points. First, keywords alone cannot retrieve images effectively because words may have varied meanings or senses. This is called the *word-aliasing* problem. Second, the number of labeled instances that can be collected from a user is limited. Through three feedback iterations, it is possible to gather just $16 \times 3 = 48$ training instances, whereas the feature dimension of this dataset is more than one hundred. Since most users would not be willing to give more than three iterations of feedback,



Semantic Modeling and Knowledge Representation for Multimedia Data. Figure 3. Cat query after iteration #3.

the system encounters the problem of scarcity of training data. Third, the negatives outnumber the relevant or positive instances being clicked on. This is known as the problem of imbalanced training data. Besides, there are a large number of images in the repository. To achieve real-time performance in query refinement and in search, efficiently indexing schemes are needed to reduce search space.

Future Directions

Major advancements in three areas are necessary before large-scale multimedia systems can be realistic: *accurate and efficient object segmentation, scalable statistical learning, and high-dimensional indexing*. For details please consult the section of Foundations.

Cross-references

► Nearest Neighbor Query in Spatio-temporal Databases

Recommended Reading

1. Aizerman M.A., Braverman E.M., and Rozonoer L.I. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.
2. Barnard K. and Forsyth D. Learning the semantics of words and pictures. *Int. Conf. on Computer Vision*, 2:408–415, 2000.
3. Beyer K., Goldstein J., Ramakrishnan R., and Shaft U. When is Nearest Neighbor meaningful. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 217–235.
4. Blei D.M., Ng A., and Jordan M. Latent Dirichlet allocation. *J. Machine Learning Res.* 3, 2003.
5. Chang E.Y. et al. Parallelizing support vector machines on distributed computers. In Proc. Advances in Neural Inf. Proc. Syst. 20, Proc. 21st Annual Conf. on Neural Inf. Proc. Syst., 2007.
6. Datta R., Joshi D., Li J., and Wang J.Z. Image retrieval: ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(65), 2008.
7. Donoho D.L. Aide-Memoire. High-Dimensional Data Analysis: The Curses and Blessings of Dimensionality (American Math. Society Lecture). In Math Challenges of the 21st Century Conference, 2000.
8. Flickner M. et al. Query by Image and Video Content: QBIC system. *IEEE Comput.*, (28), 1995.
9. Goh K., Chang E.Y., and Lai W.-C. Concept-dependent multimodal active learning for image retrieval. In Proc. 12th ACM Int. Conf. on Multimedia, 2004, pp. 564–571.
10. Hoi C.-H. and Lyu M.R. A novel log-based relevance feedback technique in content-based image retrieval. In Proc. 12th ACM Int. Conf. on Multimedia, 2004, pp. 24–31.
11. Li B. and Chang E.Y. Discovery of a perceptual distance function for measuring image similarity. *ACM Multimedia Syst.* (Special Issue on Content-Based Image Retrieval), 8(6):512–522, 2003.
12. Rocchio J.J. Relevance feedback in information retrieval. In *The SMART Retrieval System – Experiments in Automatic Document Processing*, G. Salton (ed.). Prentice-Hall, Chapter 14, 1971, pp. 313–323.
13. Rui Y., Huang T.S., and Chang S.-F. Image retrieval: current techniques, promising directions and open issues. *J. Visual Commun. Image Representation*, 1999.
14. Tong S. and Chang E.Y. Support vector machine active learning for image retrieval. In Proc. 9th ACM Int. Conf. on Multimedia, 2001, pp. 107–118.
15. Tversky A. Features of similarity. *Psychol. Rev.*, 84:327–352, 1997.
16. Wu G. and Chang E.Y. KBA: Kernel Boundary Alignment considering imbalanced data distribution. *IEEE Trans. Knowl. Data Eng.*, 17(6):786–795, 2005.

Semantic Modeling for Geographic Information Systems

CHRISTINE PARENT¹, STEFANO SPACCAPIETRA²,

ESTEBAN ZIMÁNYI³

¹University of Lausanne, Lausanne, Switzerland

²EPFL, Lausanne, Switzerland

³Free University of Brussels, Brussels, Belgium

Synonyms

Conceptual modeling; Geographical databases; GIS; Conceptual modeling for Geographic Information System; Conceptual modeling for Spatio-temporal applications

Definition

Semantic modeling denotes the activity of designing and describing the structure of a data set using a *semantic data model*. Semantic data models (also known as *conceptual data models*) are data models whose aim is to provide designers with modeling constructs and rules that are well suited for representing the user's perception of data in the application world, abstracting from implementation concerns. They contrast with *logical* and *physical data models*, whose aim is to organize data in a way that is easily manageable by a computer. The most popular semantic data models are UML, a de facto standard, and ER (Entity-Relationship), still widely used in many design methodologies and favored by the academic community.

Semantic models were first created in the database community in the 1980s. They started to be developed for Geographical Information Systems (GIS) in the 1990s. Their aim is the same as for traditional databases, to free GIS users from the specificities of system-oriented data models and proprietary file formats (e.g., spaghetti and topological data models, triangulated irregular network (TIN) models, shape files, and raster models). While a number of semantic data models for geographic data have been developed, rapidly the focus has shifted from supporting spatial data to supporting data with both spatial and temporal features, leading to the development of several spatiotemporal semantic data models. Despite the fact that current GIS and Database Management Systems (DBMS) provide poor support for temporal features, semantic modeling advocates that space and time aspects are intrinsically correlated in the application world.

Historical Background

Most people consider the 1976 paper by Peter Chen [5], defining the basic ideas of Entity-Relationship modeling, as the foundational milestone for semantic modeling. The paper had indeed an enormous effect on the database design community, leading to considerable developments to further extend the semantic capabilities of the approach. It took more than 15 years to see the same idea spreading in the academic GIS community with, for example, the 1993 MODUL-R formalism [4], which extended with spatial data the ER approach used in the leading French design methodology, Merise. Further work on MODUL-R eventually resulted in the Perceptory UML-based approach and tool [3]. Semantic models for GIS bloomed in the 1990s, basically splitting into approaches stemming from the object-oriented paradigm (e.g., [6,16]) and approaches following the ER or the UML paradigm (e.g., MADS [10], STER [15], GeoUML [2]). A survey of many spatial data models may be found in [12]. The industrial and application world has also developed GIS data modeling specifications to help promoting interoperability between different systems and different applications. The Open Geospatial Consortium (OGC) and the International Standards Organization (ISO) have produced specifications supporting conceptual modeling for data with spatial (and some temporal) features (<http://www.opengeospatial.org/>).

Thanks to the development of ubiquitous and mobile computing on the one hand, and of sensors

and GPS technologies on the other hand, large-scale capture of the evolving position of mobile objects has become technically and economically feasible. This opened new perspectives for a large number of applications (e.g., from transportation and logistics to ecology and anthropology) built on the knowledge of objects' movements. Typical examples of moving objects include cars, persons, and planes equipped with a GPS device, animals bearing a transmitter whose signals are captured by satellites, and parcels tagged with RFIDs. This fostered the interest in spatiotemporal models, rather than purely spatial or purely temporal models at the logical and semantic levels. Gütting's approach [8] defined a set of data types and associated operators for moving objects (points and surfaces), which allows one to record, for example, the changing geometry of pollution clouds and flooding waters. At the semantic level, examples of spatiotemporal models include MADS [10], Perceptory [3], STUML [15], STER [15], and ST USM [9]. Extending the limited capabilities of commercial data management systems, some research prototype systems [1,11] do provide nowadays support for storing and querying the position of a moving object all along the lifespan of the object. The latest developments in this domain are the management of trajectories, which adds a semantic interpretation to the movement of objects of kind moving point [13]. Trajectory management is important in many application domains, e.g., for addressing traffic management issues, building social models of people's movements within a city, and optimizing the localization of resources (e.g., communication antennas, shops, advertisement panels) that have to be available to moving customers.

Foundations

Requirements for Semantic Modeling of Spatial Data

Semantic modeling of spatial data requires concepts for the description of both the discrete and the continuous view of space, in a seamlessly integrated way. The *discrete view* (or *object-based view*) is the one that sees space as filled by objects with a defined location and shape. Parts of space where no object is located are considered as empty. This view typically serves application requests asking where certain objects are located, or which objects are located in a given surface. On the other hand, the *continuous view* (or *field-based view*) is the one that sees space as a continuum, holding

properties whose values depend on the location in space but not on any specific object (i.e., the value for the property is given by a function whose domain is a spatial extent). Typical examples where this view applies are the recording of continuous phenomena such as temperature, altitude, soil coverage, etc. Both views are important for applications, which may use one or the other, or both simultaneously.

Assuming the discrete view, any traditional database schema can be enriched to become a spatiotemporal database schema by including the description of the spatial and/or temporal properties of the real-world phenomena represented in the schema. Consider, for instance, a Building object type, with properties name, address, usage, architect, and owner. Adding positional information on the geographic location of the building (e.g., its coordinates in some spatial reference system) turns Building into a spatial object type. If one adds information characterizing the existence of the building in time (e.g., when construction was first decided, when construction started, when it was completed, when it was abandoned, and when it was demolished), Building becomes a temporal object type. Space and time are independent dimensions. Some data may have spatial features, some may have temporal features, some may have both, and some may have none.

Objects, be they spatial or not, can have spatial properties, i.e., properties whose value domain is composed of spatial values rather than alphanumeric values. Spatial values conform to spatial data types (see the entry in this encyclopedia), e.g., point, line, polyline, surface. For example, a Building object type can have a property nearestFireStation whose value for each building is the geographic location of the nearest fire station, e.g., a spatial value composed of two spatial coordinates defining a point.

Most basic types for space are Point, Line, and Surface (and volume for 3D databases). However, applications may require more than simple spatial data types. Some spatial objects have extents (the term “extent” denotes the set of points that an object occupies in space) that are made up of a set of elementary extents. For example, an archipelago is a set of surfaces; many coastal countries do have islands too; and facility networks may be represented by connected sets of lines. Moreover, some spatial objects have complex extents made up of a heterogeneous set of spatial values. For example, an avalanche zone is described by

a surface and a set of oriented lines describing, respectively, its maximal extent and the usual avalanche paths. Similarly, a river may be described by lines when its bed is narrow and by surfaces when it is broad. Therefore, the set of spatial data types should include types for homogeneous or heterogeneous collections, like PointSet, LineSet, SurfaceSet, or Spatial-HeterogeneousSet. The whole set of spatial data types is organized into a generalization hierarchy with generic data types, in order to support spatial object types whose extent may be of different types depending on the instance. For example, the object type City may contain large cities represented by a surface and small ones represented by a point. The spatial extent of City could then be described by a generic spatial data type that would contain points and surfaces. The Open Geospatial Consortium (OGC) has defined such a hierarchy of spatial data types.

Geographical applications often need to enforce spatial or temporal constraints between spatial or temporal features. For example, harbors should be located along water bodies and bridges on roads or railways. Therefore, a spatial data model should support constructs allowing designers to specify constraints that will be automatically enforced by the system. A first kind of construct is the spatial (or temporal) relationship type. They link two spatial (and/or temporal) object types and bear a spatial (and/or temporal) condition that the linked objects must obey. Typically, conditions express topological relationships (e.g., inclusion, disjointedness, overlapping), metric relationships (e.g., based on distance), orientation relationships (e.g., North of), or the temporal predicates defined by Allen (e.g., during). Applications may need two different kinds of these spatial and temporal relationships:

- Spatially/temporally constraining relationship types: Users can link two spatial objects by a spatially constraining relationship only if their spatial/temporal extents abide by the condition.
- Derived spatial/temporal relationship types: The system automatically creates the instances of the relationship for all couples of objects that satisfy the condition.

Moving and deforming objects may also be linked by spatial relationships. For instance, an aeronautic database may need recording the trajectories of planes when they cross storms, the two being moving objects. In these cases, the condition of the

relationship type is spatiotemporal: It bears on the location and the time.

Applications may also need constraints between composite and component elements. For example, a spatial aggregation relationship may enforce that the extent of the composite object is made up by the union of the extents of the component objects, as in a spatial aggregation linking the spatial object types Country and District. Another example is restricting the spatial (or temporal) values of attributes to be within the spatial (or temporal) extent of the object to which they belong. For example, the values of the spatial attribute major Cities (a multivalued attribute of type Point) of the object type Country should be within the spatial extent of the country. This kind of constraint may be frequent, but it is not always the case. Refer for example to the Building spatial object type with the spatial attribute nearestFireStation. Therefore, the data model should not automatically and implicitly enforce these constraints. It should provide designers with a means for explicitly specifying which constraints should be enforced.

The modeling of the continuous view of space requires another construct for properties that are defined on a spatial extent and whose value depends on the exact location (point) of the spatial extent. The spatial extent may be the whole space covered by the database or a specific extent. For example, the water quality of a river exists only in the spatial extent of the river course. On the other hand, temperature, soil, and land coverage are information that exist and may be measured (if relevant) at any point of the geographical space covered by the database. *Field-based* models are well suited for applications that perceive the real world exclusively through varying properties. For the many applications that use both the discrete and continuous views, several spatial data models provide a predominant discrete view (i.e., based on spatial objects) in addition to a special construct for representing varying properties, the *space-varying attribute*, which is a function from a spatial extent to a range of values. Any object and relationship, be it spatial or not, should be able to bear space-varying attributes. Moreover, the range of space-varying attributes may be simple (e.g., elevation) or complex (e.g., weather composed of temperature, pressure, and rainfall), monovalued (e.g., altitude) or multivalued (e.g., insects in forests, assuming this information is captured using a space unit large enough to be the home of several kinds of insect, e.g., using cells of 1 m^2).

Another important requirement for space modeling is the ability to describe data at different granularity or resolution, for example to be able to support applications working with maps at different scales.

Finally, an essential requirement is the ability to model spatial features of a phenomenon irrespectively of the fact that the phenomenon has been modeled as an object, a relationship, or an attribute. This orthogonality of the space modeling dimension with the data structure modeling dimension is what avoids making the designs in the two dimensions dependent on each other.

Survey of Current Semantic Modeling Approaches

Semantic models are typically developed in the academic world. For example, MADS [10] has been purposely developed to match all the requirements discussed in the previous section. MADS belongs to the extended ER family of models. Its distinguishing feature is the full support for multiple perceptions and multiple representations of the same real-world objects. Another distinguishing feature of MADS is its support of explicit relationships equipped with topological and synchronization constraints. Multiple perceptions and representations are also supported, to a more limited extent, by Perceptory [3], an UML extension targeted to support spatiotemporal analysis in a data-warehousing framework. STUML [15] and GeoUML [2] are other UML-based approaches, although without multi-perception support. Other spatiotemporal approaches include ST USM [9], very similar to MADS but emphasizing support of multi-granularity, and STER [15], another extended ER formalism which supports both valid and transaction time but, compared to MADS, is weaker in data structures. Most of these academic proposals have been implemented in prototypes, but, with the exception of Perceptory, they have not yet turned in commercial products. These proposals deal with 2D data.

A very different approach, known as spatial constraint database modeling, relies on mathematical equations to define spatial extents. Some existing prototype systems (e.g., DEDALE [7]) use this approach.

Key Applications

Semantic modeling is an essential capability for organizations that need to develop a database that provides different applications and different categories of users with different sets of data, possibly organized in

different ways. Designing a database in such a complex environment is a very challenging task, as has been extensively proven in traditional data management. Adding spatial features makes the design task even more complex, in particular since this inevitably leads to adding also temporal features. Indeed, what most applications in the geographical domain need to analyze is the temporal evolution of the spatial features of interest. Cartographic applications are the most traditional ones, but today the focus is rather on all kinds of planning and forecasting services to citizens and the society at the municipal, regional, and statewide levels. Examples of such services include environmental control management and global warming. Given the cost of developing databases for these applications and the need for people in charge (politicians and managers) to be successful, it is of the highest importance that the design of an operational database is carried out using the most suitable tools. Semantic modeling is the key to a successful design that determines what data are needed, to be complemented afterwards in the implementation phase by addressing performance aspects in order to guarantee that the data can be used effectively.

Semantic modeling is also the key to all data exchange, reuse, and integration efforts. Whether in database terms, as discussed here, or in ontological terms, semantic modeling is the kernel of the semantic web.

Future Directions

The economic trend towards worldwide enterprise operation and the technical trend towards web-based interoperability will significantly increase the complexity of GIS and the challenges designers will have to overcome. A key help in this context will come from ontologies about spatiotemporal application domains. These ontologies provide a common semantic basis to build repositories of domain knowledge that go beyond traditional enterprise boundaries. In this perspective, the current focus on ontology-assisted semantic modeling and ontology-assisted data integration is leading research into a fruitful direction [14].

In a complementary effort, ontologies and geographic markup languages facilitate the integration of geographical knowledge coming from multiple sources available through the Web. This will contribute to significantly enhance geographical knowledge, benefiting from geo-content actually hidden in Web pages.

Cross-references

- [Data Models](#)
- [Database Design](#)
- [Field-Based Spatial Modeling](#)
- [Geographic Information System](#)
- [Multiple Representation Modeling](#)
- [Spatial Data Types](#)
- [Topological Data Models](#)
- [Topological Relationships](#)

Recommended Reading

1. Almeida V.T., Gütting R.H., and Behr T. Querying moving objects in SECONDO. In Proc. 7th Int. Conf. on Mobile Data Management, 2006, pp. 47–51.
2. Belussi A., Negri M., and Pelagatti G. GeoUML: A geographic conceptual model defined through specialization of ISO TC211 standards. In Proc. Tenth EC GI & GIS Workshop, ESDI State of the Art, 2004.
3. Brodeur J., Bédard Y., and Proulx M.J. Modelling geospatial application database using UML-based repositories aligned with international standards in geomatics. In Proc. 8th ACM Symp. on Advances in Geographic Inf. Syst., 2000, pp. 39–46.
4. Caron C. and Bédard Y. Extending the individual formalism for a more complete modeling of urban spatially referenced data. Comput. Environ. Urban Syst., 17(4):337–346, 1993.
5. Chen P.P. The entity-relationship model: toward a unified view of data. ACM Trans. Database Syst., 1(1):9–36, 1976.
6. Egenhofer M.J. and Frank A.U. Object-oriented modeling for GIS. J. Urban Reg. Inf. Syst. Assoc., 4(2):3–19, 1992.
7. Grumbach S., Rigaux P., Scholl M., and Segoufin L. The DEDALE prototype. In Constraint Databases, Springer, 2000, pp. 365–382.
8. Gütting R.H., Böhnen M.H., Erwig M., Jensen C.S., Lorentzos N.A., Schneider M., and Vaziriannis M. A foundation for representing and querying moving objects. ACM Trans. Database Syst., 25(1):1–42, 2000.
9. Khatri V., Ram S., and Snodgrass R.T. On augmenting database design-support environments to capture the geo-spatiotemporal data semantics. Inf. Syst., 31(2):98–133, 2006.
10. Parent C., Spaccapietra S., and Zimányi E. Conceptual Modeling for Traditional and Spatio-Temporal Applications: The MADS Approach. Springer, 2006.
11. Pelekis N., Theodoridis Y., Vosinakis S., and Panayiotopoulos T. Hermes – A framework for location-based data management. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 1130–1134.
12. Rios Viqueira J.R., Lorentzos N.A., and Brisaboa N.R. Survey on spatial data modelling approaches. In Spatial Databases: Technologies, Techniques and Trends, Y. Manolopoulos, A. Papadopoulos, M. Vassilakopoulos (eds.). Idea Group, 2005, pp. 1–22.
13. Spaccapietra S., Parent C., Damiani M.L., Macedo J., Porto F., and Vangenot C. A conceptual view on trajectories. Data Knowl. Eng., 65(1):126–146, 2008.

14. Sugumaran V. and Storey V.C. The role of domain ontologies in database design: an ontology management and conceptual modeling environment. *ACM Trans. Database Syst.*, 31(3):1064–1094, 2006.
15. Tryfona N., Price R., and Jensen C.S. Spatiotemporal conceptual modeling. vol. 2520, In *Spatiotemporal Databases: The Chorochronos Approach* (chapter 3), Lecture Notes in Computer Science, vol. 2520, 2003, pp. 79–116.
16. Worboys M., Hearnshaw H., and Maguire D. Object-oriented data modelling for spatial databases. *Int. J. Geogr. Inf. Syst.*, 4(4):369–383, 1990.

Semantic Overlay Networks

GEORGE ANADOTIS, SPYROS KOTOULAS, RONNY SIEBES
VU University Amsterdam, Amsterdam,
The Netherlands

Synonyms

Semantic overlays; SONs

Definition

Semantic Overlay Networks are types of *Overlay Networks* where the topology is formed according to the resources (i.e., services or data provided) of the participants. This is done on the basis of similarity between participants, which is calculated by means of resource metadata exchange (e.g. keywords, term vectors, concepts from ontologies, histograms).

Historical Background

Semantic Overlay Networks were introduced in 2002, by H. Garcia-Molina and A. Crespo [3]. The motivation was to give an alternative to inefficient search methods for overlay networks, one that would provide more relevant results in less time and using less resources (mainly bandwidth).

Foundations

The original idea of Semantic Overlay Networks [3] was to organize the nodes that participate in a network into many different overlays, based on the content that the nodes are contributing and some classification scheme for this content. As one overlay is created for each class, the classification to be used is rather important for the operation of the system. Having classifications whose distribution is skewed would result in overlays that are not efficient (either too big or too small to be of real value). Classification as used in [3] entails hierarchy, so it is in fact a taxonomy.

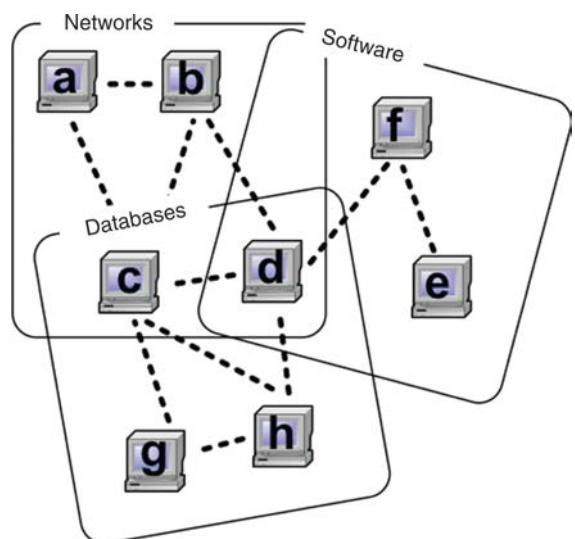
After choosing an appropriate taxonomy, content needs to be classified (either manually or automatically). This is performed massively for all content belonging to incoming nodes, so then the results of content classification are used to assign nodes to overlays, by choosing the one(s) that is the best match according to the majority of the content's classification. Different techniques and policies can be applied here (for example the predecessors and the descendants of a class are also considered), and nodes can be assigned to more than one overlay.

Finally, in order to be able to answer queries, queries are also subjected to the same classification. When they have been classified, they are subsequently forwarded only to the appropriate overlay. Results can be incremental, meaning that the query can also be sent to overlays that correspond to predecessors of the query's assigned class (i.e., more general classes) in order to retrieve additional matches.

In Fig. 1, one can see an example of three semantic overlay networks, as described above. Each node may belong to more than one semantic overlay (e.g., node d belongs to all semantic overlays) and has to maintain a number of connections for each one of them.

Inter and intra overlay routing (i.e., finding the appropriate overlay and routing within the overlay) presents a series of challenges which, along with other optimizations, have been elaborated by recent research.

In [6], the metadata used to extract similarity are (schema-less) XML documents extracted from peer



Semantic Overlay Networks. Figure 1. Three overlay networks.

content, and supported queries are extended from keyword to path queries that exploit the structure of XML documents. Peers maintain specialized data structures that summarize large collections of documents (filters – specialized extensions of Bloom filters). Each node maintains a local filter that summarizes the documents stored locally. Nodes are organized in hierarchies (trees) based on similarity of local filters; non-leaf nodes also contain merged filters summarizing the documents of its children, or in the case of root nodes, other root nodes as well.

With this organization, nodes belonging to the top levels receive more load and responsibilities, thus, the most stable and powerful nodes should be located to the top levels of the hierarchies. When receiving a query, nodes use their local filter to find results and then forward the query to their sub-tree.

In [12], the notion of *peer schemas* is defined. Peer schemas are virtually defined schemas that represent a peer's view of the world and are used for purposes of querying and mapping. Relations between peer schemas are called peer relations. Peers also contribute data to the system in the form of stored relationships, which correspond to the peer's local view of the data. Mappings are utilized in order to translate queries between different semantic networks. There are two types of mappings, namely mappings that relate two or more peer schemas (peer descriptions) as well as mappings that relate a stored schema to a peer schema (storage descriptions).

Sending queries only to peers that might provide answers is achieved using a two-fold approach: on the one hand, there is a query reformulation algorithm that works by combining global-as-view (GAV) and local-as-view (LAV) approaches and selectively applying unfolding and rewriting techniques to the original query. Since however the algorithm is only able to exploit information pertaining to schema mappings and not actual data stored at the peers, a (centralized) index structure that allows simple value lookup with partial match over structured attributes is also used. Participating peers upload data summaries as well as peer mappings to the index, thus enabling the index engine to correlate attributes from different peers and provide a simple type of schema mapping.

In [9], a variation on the approach presented in [6] is given. Instead of schema-less XML documents, metadata consist of RDF statements abiding to different RDFS schemas. There are normal peers (P) and

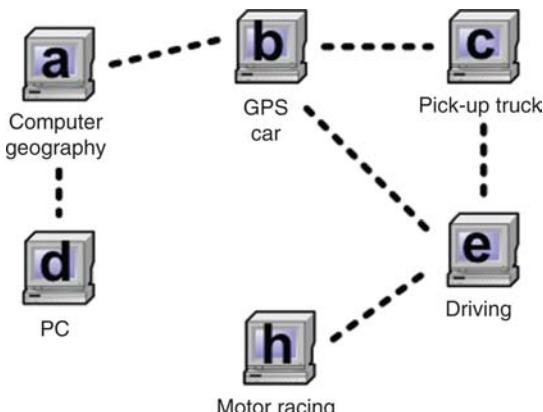
super peers (SP) that normal peers attach to, as well as index structures (SP/P and SP/SP) that utilize frequency statistics used to define similarity measures responsible for clustering peers to super-peers, thus making them dynamic. Furthermore, an efficient topology is maintained for communication in the super peer network (HyperCup), which also offers mediation services between different peer schemas.

In [2], a layered architecture for SONs is proposed, comprising of a knowledge infrastructure layer and a communication infrastructure layer. Here fully blown ontologies are used instead of taxonomies or schemas. Each peer is able to store data as well as a peer ontology. When a peer sends a query Q over the H3 network, the request goes through a query processing module for rewriting in terms of the ontological description of target concept(s). The rewritten query is then forwarded to a semantic routing module that sends it only to peers that may provide results semantically related to the kind of concepts requested (semantic neighbors).

In order to choose the semantic neighbors, semantic routing exploits the services of a knowledge manager module to retrieve ontology location links to the peers whose contents are semantically related to the target concept(s) in the query. Location links are returned to the semantic routing module, which uses them for query routing. When a peer's routing module receives a query, it forwards it to the query processing manager, where it is analyzed and processed. If no matching concepts are found, the query is discarded and no reply is returned. Otherwise the query answer is composed and forwarded to the semantic routing module which sends back the reply to the requesting peer.

In some systems, SONs are not discretely clustered and are formed on per-peer basis. While in the original approach by [3] there is a separate SON for each concept, a category of systems use pair-wise peer interactions. They are used to construct a topology enriched with content information about every neighbor, and dynamically determine routing according to this information, instead of broadcasting on the entire SON.

In [5,10], the SON is built using *advertising*. Peers exchange descriptions of their content with their neighbors. Furthermore, they keep advertisements that are similar to theirs, and thus, a semantic topology is formed. An example of such a topology is given in Fig. 2. Peers keep neighborhood relations based on their content descriptions. Thus, locality is improved. Furthermore, each peer is aware of the description of the



Semantic Overlay Networks. Figure 2. Peer neighborhood relations.

content of its neighbors. Thus, it can forward queries to the peers which are more likely to have relevant content. Research in the context of [10] indicates that either maintaining neighbor relations according to description similarity or forwarding queries to the peers with description most similar to the query perform much better than flooding approaches.

It is important to note a category of systems that use both a semantic and a structured overlay. P-Search [11] uses Latent Semantic Indexing *LSI* to extract vector representations of documents. These vectors are mapped into a multi-dimensional space maintained by a Content Addressable Network (CAN), a type of structured overlay. Nodes within this overlay participate in an additional semantic overlay used for content-directed search.

Gridvine [1] uses semantic overlays to store, query and make mappings between RDFS schemas. They are split into triples and are indexed by subject, predicate and object using the P-Grid structured overlay. Each peer may define and use its own RDFS schema, so naturally incompatibilities in terms of resource description/query interpretation may appear. To cope with such incompatibilities, the notion of schema translations is introduced. A schema translation is a mapping between two different schemas. Since RDFS does not support schema mapping, these translations are encoded using OWL.

This is a very powerful feature, as it enables the gradual forwarding of requests from the originating peer to peers for which no direct schema translation exists. This is achieved through a procedure called Semantic Gossiping, in which each peer that receives

a query expressed in a certain schema examines available translation links and evaluates (by performing syntactic and semantic analysis) if and where it should forward this query. In addition, schema inheritance is also supported, which enables peers to not only define their own schemas, but also either directly reuse or extend existing schema hierarchies. Sets of peers that share the same schema are called semantic neighborhoods.

Finally, other state-of-the-art features in SONs include observing past queries submitted/answers received in order to judge semantic proximity [13], proactively acquiring semantic links through gossiping [14], and applying ant-colony heuristics to improve semantic routing [8].

Key Applications

Semantic overlay networks use an order of magnitude less messages than flooding overlays. They enable Internet-scale systems, by providing the infrastructure for efficient search over large numbers of hosts. Despite keen interest in the area by the scientific community, none of the aforementioned systems has been commercially deployed yet.

The OpenKnowledge project (<http://www.openknowledge.org>) works on a P2P system where web-services and workflows, annotated by keywords, can be shared. The system uses a SON to store and retrieve them efficiently and to find peers that execute the webservices.

Bibster [5] is a P2P application for sharing bibliographic items. These items are annotated by concepts from an ontology. The SON is used to route queries, which are also concepts from the same ontology, to the peers that semantically match the query.

Another proposed application of SONs is using them to cluster content providers on the WWW in order to facilitate a comprehensive distributed search engine [4].

Future Directions

An interesting problem concerns optimizing multi-attribute search. Simply joining the results of single attributes is often not scalable, especially when the single attributes individually result in many answers.

Another topic is to do efficient information retrieval by using SONs. Currently, most resource discovery systems do not rank the results according to the relevance: either they match or they do not match.

Current peer-to-peer discovery systems using SONs fail to solve privacy infringement issues and are

actually more vulnerable than centralized approaches. This is because, due to the nature of SONs, peers “know” about the content of other peers, which may be undesirable.

Distributed reasoning over a P2P network is another interesting topic where SONs may be of help, for example to cluster consistent parts of knowledge. Especially in the Semantic Web area, there is a desire to have an efficient (RDF) triple storage where reasoning is done via shared or local schema's. Some first solutions like Unistore (<http://www.p-grid.org/publications/applications.html>) and Gridvine [1] are based on storing the triples in *DHTs*, which leads to many messages because each triple leads at least to three *DHT* storage- or lookup messages.

Cross-references

- ▶ Data Semantics
- ▶ DHT
- ▶ GAV
- ▶ LAV
- ▶ LSI
- ▶ Peer-to-Peer Overlay

Recommended Reading

1. Aberer K., Cudré-Mauroux P., Hauswirth M., and Pelt T.V. GridVine: Building Internet-Scale Semantic Overlay Networks. In McIlraith et al. [7], pp. 107–121.
2. Castano S., Ferrara A., Montanelli S., Pagani E., and Rossi G. Ontology-addressable contents in p2p networks. In Proc. First Workshop on Semantics in Peer-to-Peer and Grid Computing, 2003.
3. Crespo A. and Garcia-Molina H. Semantic Overlay Networks for P2P Systems, Technical Report 2003–75, Stanford University, InfoLab, 2003.
4. Doulkeridis C., Nørvåg K., and Vazirgiannis M. DESENT: Decentralized and Distributed Semantic Overlay Generation in P2P Networks, IEEE Jour. Selected Areas in Commun., 25 (1):25–34, 2007.
5. Haase P., Broekstra J., Ehrig M., Menken M., Mika P., Olko M., Plechawski M., Pyszlak P., Schnizler B., Siebes R., Staab S., and Tempich C. Bibster - A Semantics-Based Bibliographic Peer-to-Peer System. In McIlraith et al. [7], pp. 122–136.
6. Koloniari G. and Pitoura E. Content-Based Routing of Path Queries in Peer-to-Peer Systems. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 29–47.
7. McIlraith S.A., Plexousakis D., and van Harmelen F. (eds.), The Semantic Web. In Proc. 3rd Int. Semantic Web Conf., 2004.
8. Michlmayr E., Pany A., and Kappel G. Using Taxonomies for Content-based Routing with Ants. In Proc. Workshop on Innovations in Web Infrastructure, 2006.

9. Nejdl W., Wolpers M., Siberski W., Schmitz C., Schlosser M., Brunkhorst I., and Löser A. Super-Peer-Based Routing and Clustering Strategies for RDF-Based Peer-To-Peer Networks. In Proc. 12th Int. World Wide Web Conf., 2003.
10. Siebes R. and Kotoulas S. pRoute: Peer selection using shared term similarity matrices. Web Intelligence and Agent Systems, 5(1):89–107, 2007.
11. Tang C., Xu Z., and Dwarkadas S. Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks. Tech. rep., HP Labs, 2002.
12. Tatarinov I., Ives Z., Madhavan J., Halevy A., Suciu D., Dalvi N., Dong X.L., Kadiyska Y., Miklau G., and Mork P. The Piazza peer data management project. ACM SIGMOD Rec., 32(3):47–52, 2003.
13. Tempich C., Staab S., and Wranik A. Remindin': semantic query routing in peer-to-peer networks based on social metaphors. In Proc. 12th Int. World Wide Web Conf., 2004, pp. 640–649.
14. Voulgaris S., Kermarrec A.M., Massoulie L., and Van Steen M. Exploiting Semantic Proximity in Peer-to-peer Content Searching. In Proc. 10th Int. Workshop on Future Trends in Distributed Computing Systems, 2004.

Semantic Overlays

- ▶ Semantic Overlay Networks

Semantic Web

GRIGORIS ANTONIOU^{1,2}, DIMITRIS PLEXOUSAKIS^{1,2}

¹Foundation for Research and Technology-Hellas (FORTH), Heraklion, Greece

²University of Crete, Heraklion, Greece

Definition

The central idea of the Semantic Web initiative is to enrich Web content by machine-processable semantics. The approach is based on the following ideas:

1. Use meta-data (data about data) as semantic annotations
2. Use ontologies to describe knowledge needed to understand collections of Web information. The semantic annotations are linked to such ontologies
3. Use logic-based techniques to process and query collections of meta-data and ontologies

In the current Semantic Web work, two main goals can be distinguished.

Interpretation 1: The Semantic Web as the Web of Data

In the first interpretation, the main aim of the Semantic Web is to enable the integration of structured and semi-structured data sources over the Web. The main recipe is to expose data-sets on the Web enriched with semantic annotations, to use ontologies to express the intended semantics of these data-sets, in order to enable the integration and unexpected re-use of these data.

A typical use case for this version of the Semantic Web is the combination of geo-data with a set of consumer ratings for restaurants in order to provide an enriched information source.

Interpretation 2: The Semantic Web as an enrichment of the current Web

In the second interpretation, the aim of the Semantic Web is to improve the current World Wide Web. Typical use cases here are improved search engines, dynamic personalization of Web sites, and semantic enrichment of existing Web pages.

The source of the required semantic meta-data in this version of the Semantic Web is mostly claimed to come from automatic sources: concept extraction, named-entity recognition, automatic classification, etc. More recently, the insight is gaining ground that the required semantic markup can also be produced by social mechanisms of communities that provide large-scale human-produced markup.

Historical Background

The Semantic Web sprang as a vision approximately 10 years after the birth of the World-Wide Web. Not surprisingly, it was the inventor of the WWW that shaped the vision of the Semantic Web, which in turn gave rise to the entire research field.

Up until that stage, the Web was (and still is to a great extent) purely about syntax, a specific syntax geared towards homogenizing the way in which information is presented to human users via a browser. As revolutionary as the concept may have been, it was making content available only for human consumption as the interpretation of the content relied on implicit semantics. In other words, meaningful representation of content was not possible in HTML or its precursor SGML. Information and its presentation were mixed in the form of HTML documents, many of which generated automatically by applications. The Web made it easy to fetch any Web page from any server, on any platform through a uniform interface. HTML has many benefits: it is simple, textual, portable, easily searchable by keyword-based search engines

and connects pieces of information together through hypertext links. The browser is the universal application. If written properly, normal HTML markup may reflect document presentation, but it cannot adequately represent the semantics & structure of data. Newer applications require more than the publishing of HTML documents; data must be made available on the Web for use by Web-enabled applications.

XML was the incarnation of the paradigm shift on the Web: a new standard that could be easily generated and consumed by applications, facilitating data exchange across platforms and organizations, transforming the Web from a collection of documents to a collection of data published as documents. XML gained popularity very fast. It resembles HTML in that it is easy to read and learn, it is universal, portable and at the same time extensible and more flexible than HTML. However, XML cannot address all interoperability requirements as it only provides the means for solving syntactic heterogeneity problems. The challenge is to address the inherent structural but foremost semantic heterogeneities that are encountered on the Web. Modern applications need more than data on the Web; they need semantics on the Web. Applications themselves evolve into services on the Web that may exploit semantics.

The main motivation behind the Semantic Web (or Web of meaning) vision is to make vast amounts of information resources (data, documents, programs) available along with various kinds of descriptive information, i.e., metadata. Better knowledge about the meaning, usage, accessibility or quality of web resources considerably facilitates automated processing of available Web content/services especially when metadata are described in a form that is precise, human-readable and machine-interpretable. The Semantic Web enables syntactic and semantic/structural interoperability among independently-developed Web applications, allowing them to efficiently perform sophisticated tasks for humans. At the same time, it enables Web resources (data & applications) to be accessible by their meaning rather than by keywords and syntactic forms.

Foundations

The Semantic Web approach is based on the use of *semantic annotations* to describe the meaning of certain parts of Web information. For example, the Web site of a hotel could be suitably annotated to distinguish between hotel name, location, category, number of rooms, available services etc. Such metadata can

facilitate the automated processing of the information on the Web site, thus making it accessible to machines.

However, the question arises as to how the semantic annotations of different Web sites can be combined, if everyone uses terminologies of their own. The solution lies in the organization of vocabularies in so-called *ontologies*. Recommended Reading to such shared vocabularies allow interoperability between different Web resources and applications. For example, an ontology of hotel classifications in a given country could be used to relate the rating of certain hotels. And a geographic ontology could be used to determine that Crete is a Greek island and Heraklion a city on Crete. Such information would be crucial to establish a connection between a requester looking for accommodation on a Greek island, and a hotel advertisement specifying Heraklion as the hotel location.

The development of the Semantic Web proceeds in steps, each step building a layer on top of another. The basic layered design is shown in Fig. 1, which is outlined below.

1. The bottom layer comprises *XML*, a language that lets one write structured Web documents with a user-defined vocabulary. XML is particularly suitable for sending documents across the Web, thus supporting syntactic interoperability
2. *RDF* (Resource Description Framework) is a basic data model, like the entity-relationship model, for writing simple statements about Web objects (resources). The RDF data model does not rely on XML, but RDF has an XML-based syntax. Therefore it is located on top of the XML layer
3. *RDF Schema* provides modeling primitives, for organizing Web objects into hierarchies. RDF

Schema is based on RDF. RDF Schema can be viewed as a primitive language for writing ontologies

4. But there is a need for more powerful *ontology languages* that expand RDF Schema and allow the representations of more complex relationships between Web objects. Ontology languages, such as OWL (Ontology Web Language), are built on the top of RDF and RDF Schema
5. The *logic layer* is used to enhance the ontology language further and to allow writing application-specific declarative knowledge. Rule languages are the most popular logical languages used in Semantic Web applications
6. The *proof layer* involves the actual deductive process, as well as the representation and exchange of proofs in Web languages, for purposes such as explanation provision and proof validation
7. Finally, *trust* will emerge through the use of digital signatures, and other kind of knowledge, based on recommendations by agents that can be trusted, or rating and certification agencies and consumer bodies

RDF Basic Features

The language of RDF allows one to write statements. A statement consists of three parts (subject, predicate, object) and is often referred to as a triple. A triple of the form (x, P, y) corresponds to the logical formula $P(x, y)$, where the binary predicate P relates the object x to the object y ; this representation is used for translating RDF statements into a logical language ready to be processed automatically in conjunction with rules.

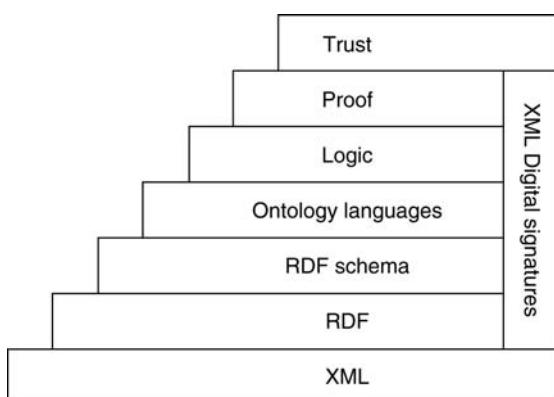
There are other ways to describe an RDF document, using a graphical and an XML representation.

RDF Schema Basic Features

In RDF, Web resources are individual objects. In RDFS, objects sharing similar characteristics are put together to form *classes*. Examples for classes are hotels, airlines, employees, rooms, excursions etc. Individuals belonging to a class are often referred to as *instances* of that class. For example, John Smith could be an instance of the class of employees of a particular hotel.

Binary *properties* are used to establish connections between classes. For example, a property *works_for* establishes a connection between employees and companies. Properties apply to individual objects (instances of the classes involved) to form RDF statements, as seen above.

The application of predicates can be restricted through the use of *domain and range restrictions*. For



Semantic Web. Figure 1. The semantic web tower.

example, the property *works_for* can be restricted to apply only to employees (domain restriction), and to have as value only companies (range restriction).

Classes can be put together in hierarchies through the *subclass relationship*: a class C is a subclass of a class D if every instance of C is also an instance of D. For example, the class of island destinations is a subclass of all destinations: every instance of an island destination (e.g., Crete) is also a destination.

The hierarchical organization of classes is important due to the notion of *inheritance*: once a class C has been declared a subclass of D, every known instance of C is *automatically* classified also as instance of D. This has far-reaching implications for matching customer preferences to service offerings. For example, a customer may wish to make holidays on an Indonesian island. On the other hand, the hotel Noosa Beach advertises its location to be Bali. It is not necessary (nor is it realistic) for the hotel to add information that it is located in Indonesia and on an island; instead, this information is inferred by the ontology automatically.

Key Applications

This section provides a bird's eye survey of key application areas. It should be noted that a healthy uptake of Semantic Web technologies is beginning to take shape in the following areas:

1. Knowledge management, mostly in intranets of large corporations
2. Data integration (Boeing, Verison and others)
3. e-Science, in particular the life-sciences
4. Convergence with Semantic Grid

If one considers the profiles of companies active in this area, they will see a distinct transition from small start-up companies such as Aduna, Ontoprise, Network Inference, Top Quadrant (to name but a few) to large vendors such as IBM (their Snobase ontology Management System}, HP (with their popular Jena RDF platform), Adobe (with their RDF-based based XMP meta-data framework), and Oracle (now lending support for RDF storage and querying in their prime database product).

However, besides the application areas listed above, there is also a noticeable lack of uptake in some other areas. In particular, promises in the areas of

1. e-commerce
2. Personalization

3. Large-scale semantic search (on the scale of the World Wide Web, not limited to intra-nets),
4. Mobility and context-awareness

are largely unfulfilled, though there is significant ongoing activity in these directions.

A pattern that seems to emerge between the successful and unsuccessful application areas is that the successful areas are all aimed at *closed communities* (employees of large corporations, scientists in a particular area), while the applications aimed at the general public are still in the laboratory phase at best. The underlying reason for this could well be the difficulty of dealing with multiple ontologies and mappings among them.

Future Directions

At present, Semantic Web research focuses, among others, on:

1. Rule languages and their interaction or integration with ontology languages (RDF and OWL)
2. Scalable storage and retrieval systems
3. Knowledge and ontology evolution and change
4. Mapping mechanisms between different ontologies

A number of items on the research agenda are hardly tackled, but do have a crucial impact on the feasibility of the Semantic Web vision. In particular:

1. The mutual interaction between machine-processable representations and the dynamics of social networks of human users
2. Mechanisms to deal with trust, reputation, integrity and provenance in a (semi-) automated way
3. Inference and query facilities that are sufficiently robust to work in the face of limited resources (be it either computation time, network latency, memory or storage space), and that can make intelligent trade-off decisions between resource use and output-quality

Cross-references

- [Interoperation of NLP-based Systems with Clinical Databases](#)
- [Ontology](#)
- [OWL: Web Ontology Language](#)
- [Resource Description Framework](#)
- [Resource Description Framework \(RDF\) Schema \(RDFS\)](#)
- [World Wide Web Consortium](#)

Recommended Reading

1. Antoniou G. and van Harmelen F. A Semantic Web Primer (2nd ed.). MIT Press Cambridge, MA, 2008.
2. Staab S. and Studer R. (eds.). Handbook on Ontologies (2nd ed.). Springer, New York, 2008.
3. Berners-Lee T., Hendler J., and Lassila O. The Semantic Web. Sci. Am., 284 (May 2001): 34–43.
4. REASE. Available at: ubp.l3s.uni-hannover.de/ubp.
5. www.SemanticWeb.org.
6. www.w3.org/2001/sw/.
7. www.ontology.org.
8. The International Semantic Web Conference (<http://iswc.semanticweb.org/>).
9. Journal of Web Semantics (www.elsevier.com/locate/websem).

Semantic Web Query Languages

JAMES BAILEY¹, FRANÇOIS BRY², TIM FURCHE²,
SEBASTIAN SCHAFFERT³

¹University of Melbourne, Melbourne, VIC, Australia

²University of Munich, Munich, Germany

³Salzburg Research, Salzburg, Austria

Synonyms

Web query languages; Ontology query languages

Definition

A number of formalisms have been proposed for representing data and meta data on the Semantic Web. In particular, RDF, Topic Maps and OWL allow one to describe relationships between data items, such as concept hierarchies and relations between the concepts. A key requirement for the Semantic Web is integrated access to data represented in any of these formalisms, as well as the ability to also access data in the formalisms of the “standard Web,” such as (X)HTML and XML. This data access is the objective of *Semantic Web query languages*. A wide range of query languages for the Semantic Web exist, ranging from (i) pure “selection languages” with only limited expressivity, to fully-fledged reasoning languages, and (ii) from query languages restricted to a certain data representation format, such as XML or RDF, to general purpose languages that support multiple data representation formats and allow simultaneous querying of data on both the standard and Semantic Web.

Historical Background

The importance of Semantic Web query languages can be traced back to the roots of the Semantic Web

itself. In its original conception, Tim Berners-Lee viewed the Semantic Web as allowing Web-based systems to take advantage of “intelligent” reasoning capabilities [4]:

- ▶ The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users For the Semantic Web to function, computers must have access to structured collections of information and sets of inference rules that they can use to conduct automated reasoning.

As the representation format for the Semantic Web has grown to cover XML, RDF, Topic Maps and OWL, there has been a corresponding growth in query languages that support access to each of these kinds of data.

Foundations

A number of techniques have been developed to facilitate powerful data retrieval on the Semantic Web. This article follows the classification and taxonomy given in [1], which provides a comprehensive survey of the area. Several categories of query languages can be distinguished, according to the format of the Semantic Web data they can retrieve:

1. Query languages for XML
2. Query languages for Topic Maps
3. Query languages for RDF
4. Query languages for OWL

XML Query Languages: Although not a primary format, it is possible to specify information on the Semantic Web using XML. Hence query languages for XML are applicable to Semantic Web data. Most query and transformation languages for XML specify the structure of the data to retrieve using either of two approaches. In the navigational approach, path-based queries over the XML data are specified and the W3C standardized languages XPath, XSLT and XQuery are well known instances of this scheme. In the example based approach, query patterns are specified as “examples” of the XML data to be retrieved. Languages of this kind are mainly research languages, with some well known representatives being XML-QL [7] and Xcerpt [3,15].

Topic Maps Query Languages: Several different query languages for Topic Maps data exist, with representatives being tolog [9], AsTMA [2] and Toma [11].

tolog was selected as the initial straw man for the ISO Topic Maps Query Language and is inspired from logic programming, also having SQL style constructs. AsTMA is a functional query language, in the style of XQuery, whereas Toma combines both SQL syntax and path expressions for querying.

RDF Query Languages can be grouped into several families, that differ in aspects such as data model, expressivity, support for schema information, and type of queries. Principal among these families is the “SPARQL Family.” This originated with the language SquishQL [12], which evolved into RDQL [12] and then was later extended to the language SPARQL [14]. These languages all “regard RDF as triple data without schema or ontology information unless explicitly included in the RDF source.” SPARQL currently has W3C Candidate Recommendation status as being the “Query Language for RDF.” In particular, SPARQL has facilities to:

1. Extract RDF subgraphs
2. Construct a new RDF graph using data from the input RDF graph queried
3. Return “descriptions” of the resources matching a query part
4. Specify optional triple or graph query patterns (i.e., data that should contribute to an answer if present in the data queried, but whose absence does not prevent an answer being returned)
5. Test the absence, or non-existence, of tuples. The general format of a SPARQL query is:

PREFIX	Specification of a name for a URI (like RDQL’s USING)
SELECT	Returns all or some of the variables bound in the WHERE clause
CONSTRUCT	Returns a RDF graph with all or some of the variable bindings
DESCRIBE	Returns a “description” of the resources found
ASK	Returns whether a query pattern matches or not
WHERE	list, i.e., conjunction of query (triple or graph) patterns
OPTIONAL	list, i.e., conjunction of optional (triple or graph) patterns
AND	boolean expression (the filter to be applied to the result)

Another family of languages for RDF, the “RQL family,” consists of the language RQL [10], and its extensions such as SeRQL [5]. Common to this family is support for the combination of both data and schema querying. The RDF data model which is used slightly deviates from the standard data model for RDF and RDFS, disallowing cycles in the subsumption hierarchy and requiring both a domain and a range to be defined for each property. RQL itself has a large number of features and choices in syntactic constructs. This results in a complex, yet powerful language, which is far more expressive than other RDF query languages, especially those of the SPARQL family.

A number of other types of query languages for RDF also exist, using alternative paradigms. These include query languages using reactive rules, such as Algae [13] and deductive languages such as TRIPLE [6] and Xcerpt [15,3]. The last of these is noteworthy, as it combines querying on both the Standard Web (HTML/XML), with querying on the Semantic Web (e.g. RDF, TopicMaps) and also allows pattern-based, incomplete specification of queries.

OWL Query Languages: Query languages for OWL are still in their infancy compared to those for RDF. OWL-QL [8] is a well known language for querying OWL data and is an updated version of the DAML Query language. Its design targets the assistance of query-answering dialogues between computational agents on the Semantic Web. Unlike the RDF query languages, it focuses on the querying of schema rather than instance data. An RDF language such as SPARQL may of course be used to query OWL data, but it is not well suited to the task, since it is not designed to be aware of OWL semantics.

Several themes emerge from considering the design of the various Semantic Web Query languages [1].

- *Choice of querying paradigm:* Semantic Web query languages express basic queries using either the path based (navigational) or logic based (positional) paradigm.
- *Choice of variable type:* When Semantic Web query languages have variables, they almost always are logical variables, as opposed to variables in imperative programming languages.
- *Provision of Referential Transparency and Answer-Closure.* Referential Transparency (i.e., within the same scope, an expression always means the same), a well known trait of declarative languages, is striven

for by Semantic Web query languages. Answer closedness is a property that allows answers to queries themselves to be used as input to queries and is a key design principle of the languages SPARQL and Xcerpt.

- *Degree of Incompleteness:* Many Semantic Web query languages offer a means for incomplete specifications of queries, a reflection of the semi-structured nature of data on the Semantic Web.
- *Reasoning Capabilities.* Interestingly, but not surprisingly, not all XML query languages have views, rules, or similar concepts allowing the specification of other forms of reasoning. Surprisingly, the same holds true of RDF query languages. Many authors of RDF query languages see deduction and reasoning to be a feature of an underlying RDF store offering materialization, i.e., completion of RDF data with derivable data prior to query evaluation. This is surprising, because one might expect many Semantic Web applications to access not only one RDF data store at one Web site, but instead many RDF data stores at different Web sites and to draw conclusions combining data from different stores.

Key Applications

Like classical query languages such as SQL, the first key application of Semantic Web query languages is the efficient and scalable access, classification, analysis and transformation of large collections of data in a Web format such as XML, RDF, OWL, or Topic Maps. Whereas classical query languages are most often used for accessing a single, centralized database, Semantic Web query languages need to be able to access also remote databases and data sources. This opens up new application scenarios, potentially utilizing any of the vast number of the data sources available on the Web.

For example, one might query researcher and publication information integrated over various sources, such as DBLP, Citeseer, IEEE and Cordis, combine that data with course and lecturer information from the Semantic Web School and then even further correlate it with the US census data. All these resources would be far too large to download individually and query locally, but they provide interfaces known as *endpoints*, that can be used to select the relevant portions via a Semantic Web query interface. Another example application is the W3C Amaya browser, which can be used to enrich Web pages visited by a user, with annotations contained in remote data sources. The annotations

relevant to a given Web page are accessed by querying an annotation server using Algae [13], an RDF query language similar to SPARQL. In such scenarios, the ability of RDF (and to some extent, XML) to define the names and concepts used in a database, reason about them and to map them to names and concepts used in another database, is essential. This clearly separates the use of Semantic Web query languages from the use of classical query languages for centralized databases.

Increasingly, current Web applications (often referred to as Web 2.0 applications) contain a JavaScript-based user interface which is separate from the data processed by the application itself. Thus, the user interface can be loaded once and data then requested from the origin server or other data sources on the Web as required. Web query languages for XML, RDF, JSON and Topic Maps are now becoming recognized as the ideal interfaces between the client user interfaces of Web 2.0 applications and data sources, since they can target just the data that is needed in the current state of the application. Web query languages allow flexible, but fine-grained access to the required data, rather than the coarse-grained access provided by other solutions.

Future Directions

Most RDF query languages are RDF-specific, and even specifically designed for one RDF serialization, which of course limits their applicability. It is to be hoped that in the future, there will be an evolution towards data format “versatile” languages, capable of easily accommodating XML, RDF, Topic Maps and OWL, without requiring “serialization consciousness” from the programmer.

The method of query evaluation in current Semantic Web query languages is either backtracking-free logic programming (as used by positional languages) or set-oriented functional query evaluation. It seems likely these two paradigms may converge in future Semantic Web query languages. Language engineering issues, such as abstract data types and static type checking, modules, polymorphism, and abstract machines, have not yet made their way into Semantic Web query languages, as they did not in database query languages. This situation opens avenues for promising research of great practical, as well as theoretical relevance.

Data Sets

There are a number of SPARQL endpoints that can be browsed on the Web. These provide RDF data

which can be viewed and then queried using a SPARQL client:

- The 2000 US Census Data endpoint: <http://www.rdfabout.com/demo/census/>
- The Semantic Web School endpoint: <http://sparql.semantic-web.at/>
- A compilation of endpoints including DBLP, Cite-seer, IEEE and Cordis: <http://www.rkbexplorer.com/>

A collection of concrete query language use cases for accessing RDF data can be found in the W3C RDF Use Case document at <http://www.w3.org/TR/rdf-dawg-uc/>. A use case collection is also included in [2].

URL to Code

The D2R Server is a utility for publishing relational databases on the Semantic Web and can be found at: <http://sites.wiwi.fu-berlin.de/suhl/bizer/d2r-server/>

Annotea is a project that aims to assist collaboration via shared semantic meta-data. The Annotea Server with Amaya Browser and Algae QL can found at: <http://www.w3.org/2001/Annotea/>

Cross-references

- ▶ Ontology
- ▶ OWL: Web Ontology Language
- ▶ Resource Description Framework
- ▶ Resource Description Framework (RDF) Schema (RDFS)
- ▶ Semantic Web
- ▶ Topic Maps
- ▶ XML
- ▶ XPath/XQuery
- ▶ XSL/XSLT

Recommended Reading

1. Bailey J., Bry F., Furche T., and Schaffert S. Web and semantic Web query languages: a survey. In Reasoning Web, LNCS 3564, Springer, 2005, pp. 35–133.
2. Barta R. AsTMa 1.3 language specification. Technical report, Bond University, 2003.
3. Berger S., Bry F., Furche T., Linse B., and Schroeder A. Beyond XML and RDF: the versatile Web query language Xcerpt. In Proc. 15th Int. World Wide Web Conf., 2006, pp. 1053–1054.
4. Berners-Lee T., Hendler J., and Lassila O. The Semantic Web—a new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. Scientific American, 2001, 29–37.

5. Broekstra J. and Kampman A. SeRQL: a second generation RDF query language. In Proc. SWAD-Europe Workshop on Semantic Web Storage and Retrieval, 2003.
6. Decker S., Sintek M., Billig A., Henze N., Dolog P., Nejdl W., Harth A., Leicher A., Busse S., Ambite J.L., Weathers M., Neumann G., and Zdun U. TRIPLE – an RDF rule language with context and use cases. In Proc. Rule Languages for Interoperability, 2005.
7. Deutsch A., Fernandez M., Florescu D., Levy A., and Suciu D. A query language for XML. Comput. Netw., 31(11–16):1155–1169, 1999.
8. Fikes R., Hayes P., and Horrocks I. OWL-QL – a language for deductive query answering on the semantic Web. J. Web Semant., 2(1):19–29, 2004.
9. Garshol L.M. Tolog – a topic maps query language. In Proc. 1st Int. Workshop on Topic Maps Research and Applications, 2005, pp. 183–196.
10. Karvounarakis G., Magkanarakis A., Alexaki S., Christophides V., Plexousakis D., Scholl M., and Tolle K. Querying the Semantic Web with RQL. Comput. Netw. ISDN Syst. J., 42(5):617–640, August 2003.
11. Lacher M. and Decker S. RDF, topic maps, and the semantic web. Markup Lang. Theory and Pract., 3(3):313–331, December 2001.
12. Miller L., Seaborne A., and Reggiori A. Three implementations of SquishQL, a simple RDF query language. In Proc. Int. Semantic Web Conf., 2002, pp. 423–435.
13. Prud'hommeaux E. Algae RDF Query Language. <http://www.w3.org/2004/05/06-Algae/>, 2004.
14. Prud'hommeaux E. and Seaborne A. SPARQL query language for RDF. Candidate recommendation, W3C, June 2007, <http://www.w3.org/TR/rdf-sparql-query/>.
15. Schaffert S. and Bry F. Querying the Web reconsidered: a practical introduction to Xcerpt. In Proc. Extreme Markup Languages, 2004.

Semantic Web Services

DAVID MARTIN

SRI International, Menlo Park, CA, USA

Definition

Semantic Web Services (SWS) is a research area developing theory, technology, standards, tools, and infrastructure for working with distributed, networked services. As its name indicates, SWS has arisen from the cross-fertilization of challenges and approaches from the Web services and Semantic Web areas. The central theme of SWS is the enrichment of Web services technology with knowledge representation and reasoning technologies

(including but not limited to those associated with the Semantic Web). The starting point for most SWS approaches is the use of expressive, declarative descriptions of the elements of dynamic distributed computation, with a particular focus on services, processes that are encapsulated by services, and message-based conversations between service providers and consumers. (Depending on the approach, other relevant concepts might include goals, transactions, roles, commitments, mediators of various kinds, etc.) These descriptions, in turn, are seen as the basis for fuller, more flexible automation of service provision and use, and the construction of more powerful components, architectures, tools and methodologies for working with services. In most cases, descriptions are expressed in a formal logical framework allowing for the use of well-understood reasoning procedures.

Many SWS researchers have articulated a broad and ambitious long-term vision of a Web where support for shared *activities* is as central as support for shared *information*. Many view SWS, developed to its full potential, as a technology foundation for distributed autonomous agents (and much SWS work draws on earlier work on agent-based systems). Another important theme in SWS is the development of a unified, comprehensive representation framework (often making use of ontologies) that can provide a foundation for a broad range of activities throughout the Web service lifecycle, including design and development, publication in registries, discovery and selection, negotiation and contracting, composition of services, monitoring and recovery from failure, and so forth.

Key Points

SWS research, as a distinct field, began in earnest in 2001. In that year, the initial release of OWL for Services (OWL-S) [5] became available. Other major initiatives began not long thereafter, including the Web Services Modeling Ontology (WSMO) [4], the Semantic Web Services Framework (SWSF) [2], WSDL-S [1], and the Internet Reasoning Service [3]. Many individual researchers and small teams have also done much valuable work, sometimes drawing on one of these larger efforts, sometimes not.

A fair amount of work in SWS has been focused on two central problems. Given a service request and a collection of service descriptions, *service discovery* is the problem of identifying those services that can

satisfy the request, and possibly ranking them according to some measure of suitability. Given a goal to be satisfied and a collection of service descriptions, *service composition* is the problem of finding a procedure composed of service invocations that will achieve that goal. It should be emphasized, however, that SWS is a broad field with many challenging problems, of which these two are mentioned as illustrations.

Important application areas for SWS have included business (e.g., automated or partially automated discovery and use of needed services, enactment and composition of business processes and workflow, supply chain management, contracting, formation of virtual organizations, etc.), e-Government, and e-Science.

A few SWS standards activities have occurred. For example, the World Wide Web Consortium (W3C) has published a set of extensions to the Web Services Description Language (WSDL), known as Semantic Annotations for WSDL (SAWSLD), which makes it possible to associate elements of WSDL specifications with elements defined in a SWS framework (not defined by SAWSLD). W3C also hosts workshops and study groups to consider the suitability of various aspects of SWS for standardization.

Cross-references

- ▶ [Semantic Web](#)
- ▶ [Web Services](#)

Recommended Reading

1. Akkiraju R., Farrell J., and Miller J., et al. Web Service Semantics – WSDL-S, vol. 1.0, tech. note, Apr. 2005.
2. Battle S., Bernstein A., and Boley H., et al. Semantic Web Services Framework (SWSF) Overview, 2005.
3. Cabral L., Domingue J., and Galizia S., et al. IRS-III: a broker for semantic web services based applications. In Proc. 5th International Semantic Web Conference, 2006, pp. 201–214.
4. Fensel D., Lausen H., and Polleres A., et al. Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer, New York, 2006.
5. Martin D., Burstein M., and McDermott D., et al. Bringing semantics to web services with OWL-S. World Wide Web J., 10(3): 243–277, 2007.

Semantic-based Retrieval

- ▶ [Multimedia Information Retrieval Model](#)

Semantic Atomicity

GREG SPEEGLE

Baylor University, Waco, TX, USA

Definition

Let T be a transaction composed of subtransactions S_0, S_1, \dots, S_{n-1} . Let C_0, C_1, \dots, C_{n-1} be a set of *compensating transactions*, such that C_i compensates for the corresponding S_i . T is *semantically atomic* iff all S_i have committed, or for all S_i that have committed, C_i has also committed. A schedule (or history) ensures semantic atomicity if all transactions are semantically atomic. If T requires compensating transactions, then the resulting database is semantically equivalent to one in which T did not execute at all, but it is not guaranteed to be identical. Typically, two database states are equivalent if they both satisfy all of the database constraints.

Historical Background

Semantic Atomicity is first defined in [6], with the use of *countersteps* to remove parts of a failed transaction executing in a distributed database environment, without rolling back the entire transaction. The “step” grew in complexity to a subtransaction with the introduction of Sagas [7]. This required a corresponding increase in the complexity of the counter-measure, now called *compensating transactions*. Within Sagas, subtransactions are allowed to interleave with other transactions, and an execution is correct if every subtransaction commits, or the corresponding compensating transaction is executed. In [10], transactions are extended to transaction programs with input and output constraints, thus allowing formal representations of the capabilities and requirements for compensating transactions. Semantic atomicity has been applied in multidatabases[4], multilevel secure databases [2], workflows [3], and real-time database systems [16].

Foundations

Under traditional database correctness criteria, either all of the updates performed by a transaction must be committed to the database, or none of them should be. This is the *atomicity* requirement in ACID (atomicity, consistency, isolation and durability), the standard correctness criteria for database transactions. Atomicity has potentially far-reaching consequences.

Consider the scenario where transaction T_1 updates some data, and transaction T_2 reads the updated, but not committed, value. If T_1 fails, then the update must be removed from the database, which implies T_2 cannot have read that value, and thus, T_2 must be aborted as well. This situation is called *cascading aborts*.

Within traditional database applications, atomicity (and consequently cascading aborts) is the correct criterion. Transactions in this environment are very short (only a few operations) and access only a few data items. In fact, the traditional database environment avoids cascading aborts by using a protocol called strict two-phase locking [8], which forces T_2 to wait until T_1 commits before it can read anything written by T_1 .

The success of database management systems and the explosion of electronic data has pushed DBMSs (database management systems) into non-traditional applications. In these applications, transactions can be very long and very complex (e.g., design applications [9] and workflows [3]). In these applications, cascading aborts are unacceptable, but it is also unacceptable to force long-duration waits with two-phase locking. There are many efforts in the literature to solve this problem by exploiting the semantic information of the transactions (see e.g., [1,5,7,11,12,17]).

These solutions focus on relaxing one or more of the ACID properties while still ensuring the correct execution of the DBMS. This relaxation can only occur by using additional information (semantics) not exploited in traditional database systems. *Semantic atomicity* focuses on relaxing the atomicity requirement by using additional information in the form of *compensating transactions* for any transaction that is executing in the system [7]. A compensating transaction uses application specific information to restore the database consistency constraint for any failed transaction.

Additionally, semantic atomicity is often used in environments where transactions are nested [14], which means transactions are divided into subtransactions. One way to prevent cascading aborts is to prohibit the exposure of the results of a subtransaction beyond the transaction itself. However, with semantic atomicity, these intermediate results can be visible to other transactions and cascading aborts can still be avoided. This is one of the benefits of the long duration transaction model called *Sagas* [7].

Semantic atomicity allows transactions to read dirty data if all failed transactions meet the following compensation requirement:

Definition: Let T be a transaction, H be the set of all transactions concurrently executing with T (excluding T), and let C be the compensating transaction for T . Let D represent the database state resulting from executing THC on the database, and D' be the database state resulting from executing H alone. A transaction has been compensated if $D \equiv D'$.

In general, two database states are equivalent if they both satisfy all database consistency constraints. Likewise, it must be the case that the database state seen by H is consistent, otherwise the execution of some of the transactions in H is unpredictable. It is also possible for subtransactions to have reduced consistency requirements, both in terms of the state of the database before transactions execute, and in the database state after the execution is finished [10]. In these cases, the compensating transaction does not have to completely restore the database state, but it must create a database state that allows the other subtransactions to continue.

An example can clarify the benefits of semantic atomicity. Consider a business process in which a company manufactures widgets. Some of the widgets are sold after they are manufactured, and others are ordered in advance. The database consistency constraint is that the number of widgets in production must be at least twice the number of widgets that have been pre-ordered. Let P and O be data items in the database such that P is the number of widgets in production, and O is the number of ordered widgets. The constraint is that $P \geq 2 \times O$. Furthermore, assume a purchase transaction, T , is composed of two parts – order placement (S_0) and payment (S_1). Finally, assume the current database state has $O = 10$ and $P = 23$.

An order comes in for 3 additional widgets. Since the company wants to produce the needed widgets as soon as possible, the database is updated right away by having transaction S_0 set O to 13, and correspondingly, increasing P to 26. Under semantic atomicity, S_0 is free to commit. As soon as P is increased, a workflow transaction W begins the manufacturing process for three additional widgets.

Now the customer decides the price for the three widgets is too high, and cancels the order. As a result, T has failed. Under traditional atomicity, the update by T would rollback, and therefore the execution of

W would also abort. Note that aborting W may not even be possible, depending on the state of widget production. Under semantic atomicity, a compensating transaction C_O is executed instead. C_O performs a rollback on the value of O , but leaves P unchanged. Therefore, W can continue execution. Thus, consistency is restored and the schedule is semantic atomic, even though W would *never execute* without T . The database states are equivalent (both are consistent), but not identical.

One great but not immediately obvious benefit of semantic atomicity is that subtransactions can commit as soon as possible, thus externalizing the effects of the subtransactions right away. This is because a later failure cannot cause a cascading abort. The appropriate compensating transaction is executed, and the transaction system continues forward. As a result, long duration transaction systems do not have to impose long duration waits to avoid cascading aborts. Thus, the two primary disadvantages for ensuring ACID with advanced transactions (long waits and large amounts of lost work) are prevented.

Unfortunately, compensating transactions cannot always be applied automatically. Consider the example from before, but assume the constraint is $2 \times O \leq P \leq 2.5 \times O$. The compensation for S_0 cannot simply rollback O , as the resulting state would be inconsistent. One possible solution would require the compensation of W , perhaps removing a widget from production. Whether or not this is possible would depend on the application semantics. Alternatively, S_0 cannot be compensated, and thus semantic atomicity reduces to traditional atomicity. In this case, T would not commit until payment is received, and the extra widgets would not begin production until the order is committed.

The complexity of using application semantics in building compensating transactions prevents the common deployment of semantic atomicity. In [10] these problems are studied in detail, with the creation of *operations* which are arbitrarily complex modifications to a single database entity. These operations are allowed local variables, thereby resembling functions in traditional programming languages. The operations are combined into *transaction programs*, which include conditional statements and statement blocks. Within this model, several aspects of compensating transactions are explored, such as compensation when the database states must be identical (not just equivalent) and

compensation when some transaction program must follow the compensated-for transaction (called unsound).

Another key issue mentioned in [10] is the requirement that compensating transactions do not fail. Although this can be ensured during normal database operations (e.g., using a deadlock avoidance mechanism for compensating transactions), system failures cannot be prevented. The solution to this problem requires logging the internal state of the compensating transaction as well as any database modifications. During recovery, incomplete compensating transactions are not aborted, but are continued from the saved internal state, similar to the notion of compensating log records in ARIES [13].

Note that semantic atomicity is distinctly different from the concept of a *savepoint*. A savepoint does not expose the updates of a transaction to outside processes (called externalized operations in [10]), while semantic atomicity supports this. Likewise, when a traditional database transaction performs a rollback to a savepoint, any other transaction which has read the aborted updates, must also abort. Thus, savepoints do not prevent cascading aborts.

Key Applications

Semantic atomicity is appropriate for any application where the benefit of avoiding cascading aborts without long-duration waits is greater than the difficulty of creating the compensating transactions. Examples include:

1. Multidatabases [4] where each site can commit a subtransaction without the overhead of two-phase commit
2. Application services such as the Microsoft Phoenix project [3] which support applications surviving database failures in part by using semantic information
3. Workflows [3] where processes need to begin as soon as possible, and often cannot be aborted
4. Web services [15] where actions are performed by loosely coupled systems
5. Real-time systems [16] where the ability to predict transaction length is greatly improved by avoiding cascading aborts and by allowing early commits
6. Secure databases [2] where cascading aborts can cause covert information exchange

Although not an application per se, support for compensating transactions, and thereby semantic atomicity, has been included in the Organization for the Advancement of Structured Information Standards

(OASIS) Web Services Business Activity (WS-Business-Activity) standard released in July 2007. Certainly, this will increase the number of commercial applications using semantic atomicity.

Cross-references

- [ACID](#)
- [Atomicity](#)
- [Compensating Transactions](#)
- [Extended Transaction Models and ACTA](#)
- [Nested Transactions](#)
- [Open-Nested Transaction Model](#)
- [Sagas](#)
- [Workflows](#)

Recommended Reading

1. Ahmed E. (ed.). Database transaction models for advanced applications. Data Management Systems. Morgan Kaufmann, Los Altos, CA, 1992.
2. Ammann P., Jajodia S., and Ray I. Ensuring atomicity of multi-level transactions. In Proc. IEEE Symp. on Research in Security and Privacy, 1996, pp. 74–84.
3. Breitbart Y., Deacon A., Schek H.-J., Sheth A., and Weikum G. Merging application-centric and data-centric approaches to support transaction-oriented multi-system workflows. ACM SIGMOD Rec., 22(3):23–30, 1993.
4. Breitbart Y., Garcia-Molina H., and Silberschatz A. Overview of multidatabase transaction management. VLDB J., 1(2):181–240, 1992.
5. Chrysanthis P.K. and Ramamritham K. Synthesis of extended transaction models using ACTA. ACM Trans. Database Syst., 19(3):450–491, 1994.
6. Garcia-Molina H. Using semantic knowledge for transaction processing in a distributed database. ACM Trans. Database Syst., 8(2):186–213, June 1983.
7. Garcia-Molina H. and Salem K. Sagas. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1987, pp. 249–259.
8. Gray J., Lorie R., Putzolu G., and Traiger I. Granularity of locks and degrees of consistency in a shared database. In Readings in Database Systems, Morgan Kaufmann, 1998, pp. 94–121.
9. Korth H.F., Kim W., and Bancilhon F. On long duration CAD transactions. Inf. Sci., 46:73–107, October 1988.
10. Korth H.F., Levy E., and Silberschatz A. A formal approach of recovery by compensating transactions. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 95–106.
11. Korth H.F. and Speegle G. Formal aspects of concurrency control in long-duration transaction systems using the NT/PV model. ACM Trans. Database Syst., 19(3):492–535, 1994.
12. Lynch N., Merritt M., Wiehl W., and Fekete A. Atomic transactions. Data Management Systems. Morgan Kaufmann, 1994.
13. Mohan C., Haderle D., Lindsay B., Pirahesh H., and Schwarz P. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging. ACM Trans. Database Syst., 17:94–162, March 1992.
14. Moss J.E.B. Nested Transactions – An Approach to Reliable Distributed Computing. MIT Press, Cambridge, MA, 1985.

15. Puustjarvi J. Using advanced transaction and workflow models in composing web services. In *Adv. Comput. Sci. Technol.*, 2007.
16. Soparkar N., Levy E., Korth H.F., and Silberschatz A. Adaptive commitment for distributed real-time transactions. In *Proc. Int. Conf. on Information and Knowledge Management*, 1994, pp. 187–194.
17. Weikum G. Principles and realization strategies of multilevel transaction management. *ACM Trans. Database Syst.*, 16(1): 132–180, 1991.

Semantics-based Concurrency Control

KRITHI RAMAMRITHAM¹, PANOS K. CHRYSANTHIS²

¹Indian Institute of Technology Bombay, Mumbai, India

²University of Pittsburgh, Pittsburgh, PA, USA

Definition

Specifications of data contain semantic information that can be exploited to increase concurrency. For example, two insert operations on a multiset object commute and hence, can be executed in parallel; further, regardless of whether one operation commits, the other can still commit. Applying the same rule, two push operations on a stack object do not commute and hence cannot be executed concurrently. Several schemes have been proposed for exploiting the semantics of operations have to provide more concurrency than obtained by the conventional classification of operations as *reads* or *writes*.

Key Points

In most semantics-based protocols, conflicts between operations is based on commutativity, an operation o_i which does not commute with other uncommitted operations will be made to wait until these conflicting operations abort or commit. Some protocols use operations' return value commutativity, wherein information about the results of executing an operation is used in determining commutativity, and some use the arguments of the operations in determining whether or not two operations commute. An example of the former, two increment operations on a counter object commute as long as they do not return the new or old value of the counter. An example of the latter, two insert operations on a set object commute as long as they do not insert the same item.

In the scheme reported in [1], non-commuting but *recoverable* operations are allowed to execute in parallel;

but the order in which the transactions invoking the operations should commit is fixed to be the order in which they are invoked. If o_j is executed after o_i , and o_j is *recoverable relative to o_i*, then, if transactions T_i and T_j that invoked o_i and o_j respectively commit, T_i should commit before T_j . Thus, based on the recoverability relationship of an operation with other operations, a transaction invoking the operation sets up a dynamic commit dependency relation between itself and other transactions. If an invoked operation is not recoverable with respect to an uncommitted operation, then the invoking transaction is made to wait. For example, two pushes on a stack do not commute, but if the push operations are forced to commit in the order they were invoked, then the execution of the two push operations is serializable in commit order. Further, if either of the transactions aborts the other can still commit.

In [2] authors make an effort to discover, from first principles, the nature of concurrency semantics inherent in objects. Towards this end, they identify the dimensions along which object and operation semantics can be modeled. These dimensions are then used to classify and unify existing semantic-based concurrency control schemes. To formalize this classification, a graph representation for objects that can be derived from the abstract specification of an object is proposed. Based on this representation, which helps to identify the semantic information inherent in an object, a methodology is presented that shows how various semantic notions applicable to concurrency control can be effectively combined to improve concurrency. A new source of semantic information, namely, the ordering among component objects, is exploited to further enhance concurrency. Lastly, the authors present a scheme, based on this methodology, for *deriving* compatibility tables for operations on objects.

Cross-references

- ACID Properties
- Concurrency Control – Traditional Approaches

Recommended Reading

1. Badrinath B.R. and Ramamritham K. Semantics-based concurrency control: beyond commutativity. *ACM Trans. Database Syst.*, 17(1):163–199, 1991.
2. Chrysanthis P.K., Raghuram S., and Ramamritham K. Extracting concurrency from objects: a methodology. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1991.

Semijoin

KAI-UWE SATTLER

Technical University of Ilmenau, Ilmenau, Germany

Synonyms

Semijoin filter; Hash filter join; Bit vector join; Bloom filter join; Bloom join

Definition

Semijoin is a technique for processing a join between two tables that are stored at different sites. The basic idea is to reduce the transfer cost by first sending only the projected join column(s) to the other site, where it is joined with the second relation. Then, all matching tuples from the second relation are sent back to the first site to compute the final join result.

Historical Background

The semijoin technique was originally developed by Bernstein et al. [3] as part of the SDD-1 project as a reduction operator for distributed query processing. The idea of applying hash filtering was proposed by Babb [1] as well as by Valduriez [9] particularly for specialized hardware (content addressed file stores and distributed database machines respectively). The theory of semijoin-based distributed query processing was presented in [2]. In [10] semijoins are also exploited for query processing on multiprocessor database machines. Results of detailed experimental work on semijoins in distributed databases were first reported by Lu and Carey [6] as well as by Mackert and Lohman [7].

Foundations

Semijoin is a join processing technique which was originally developed for distributed databases. A semijoin is the “half of a join” and is particularly useful as a reduction operator.

Relational Definition

Given two relations $R(A,B)$ and $S(C,D)$ with the join condition $R.A = S.C$ the semijoin $R \bowtie S$ is defined as follows:

$$R \bowtie_{A=C} S = \pi_{attr(R)}(R \bowtie_{A=C} S)$$

where $attr(R)$ denotes the set of attributes in R . The semijoin has two important characteristics:

1. It is a reducing operator, because $R \bowtie_{A=C} S \subseteq R$.
2. It is asymmetric, i.e., $R \bowtie_{A=C} S \neq S \bowtie_{A=C} R$.

Semijoin Filtering

The obvious approach of processing a join between a relation R stored at site 1 and S stored at site 2 is to ship the smaller relation to the other site and compute the join locally. This is also called “ship whole” approach. However, for computing the join one or both of relations can be replaced by a semijoin with the other relation, i.e.,:

$$\begin{aligned} R \bowtie_{A=B} S &= (R \bowtie_{A=C} S) \bowtie_{A=C} S \\ &= R \bowtie_{A=C} (S \bowtie_{C=A} R) \\ &= (R \bowtie_{A=C} S) \bowtie_{A=C} (S \bowtie_{C=A} R) \end{aligned}$$

In each case the semijoin acts as a reducer operation just like a selection operator. Which variant is chosen for the actual join processing has to be decided by estimating the costs.

The principled approach of the semijoin filtering can be formulated in the following algorithm:

1. At site 1 compute $R' := \pi_A(R)$ and send it to site 2
2. At site 2 process the semijoin $S' := S \bowtie_{C=A} R'$. Note, relation S' contains only tuples matching the join condition and will appear in the final result. Furthermore, the result relation provides only the attributes from S
3. Send relation S' to site 1
4. At site 1, $R \bowtie_{A=C} S'$ is computed producing a result equivalent to $R \bowtie_{A=C} S$

In Fig. 1 the process is illustrated using an example.

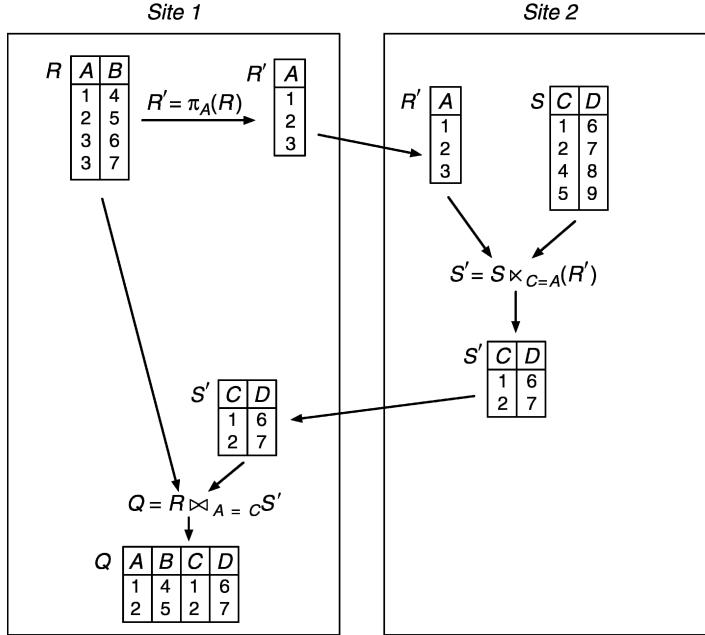
In order to estimate the benefit of the semijoin compared to the “ship whole” approach it is sufficient to consider only the transfer costs. Let C denote the cost for transfer a data unit and $size(R) = |R| \cdot width(R)$ the size of the relation derived from the cardinality $|R|$ and the size $width(R)$ of a tuple in data units. Then, the cost for the ship whole strategy is

$$C \cdot size(S)$$

assuming S is the smaller relation ($size(S) < size(R)$). For the semijoin the main costs are in step 1 and step 3, i.e.,

$$C \cdot size(\pi_A(R)) + C \cdot size(S \bowtie_{C=A} R)$$

Comparing these costs, one can observe that the semijoin approach is better if



Semijoin. Figure 1. Example of semijoin processing.

$$\text{size}(\pi_A(R)) + \text{size}(S \otimes_{C=A} R) < \text{size}(S)$$

More exactly due to $\text{width}(S \otimes_{C=A} R) = \text{width}(S)$ the semijoin is the better choice if $|S \otimes_{C=A} R| < |S|$, i.e., if the semijoin is really a reducer. At the other hand, the ship whole approach is better if nearly all tuples of S contribute to the join result. In this case, the semijoin has the disadvantage of the additional transfer of $\pi_A(R)$.

Thus, a decision for one of these join strategies requires an estimation of the join selectivity factor SF. For the semijoin the following approximation

$$SF_{R \bowtie_{A=C} S} = \frac{|\pi_C(S)|}{|\text{distinct}(C)|}$$

was proposed by [5], where $|\text{distinct}(C)|$ denotes the number of distinct values in attribute C .

Bit Vector Filtering

The effort for step 1 of the semijoin can be further reduced by sending only a compact bitmap representation of the column values instead of $\pi_A(R)$. This bitmap or bit vector is built using a hash function [4] and, thus, the approach is called bit vector filtering, hash filter join or bloom join.

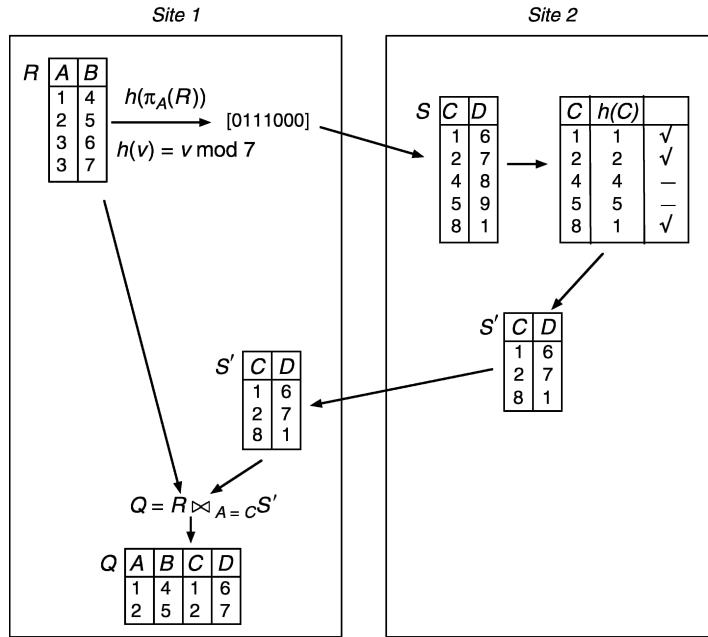
For a hash function $h(v)$ returning values $0 \dots n$ a bit vector $B[0 \dots n]$ containing $n + 1$ bits initially set to 0 is required. For each value $v \in \pi_A(R)$ the corresponding

bit $B[h(v)]$ is set to 1. Instead of processing the semijoin $S \bowtie R$ at site 2, this bit vector and the hash function are used to probe the tuples of S for matching with the join attribute B , i.e., if for a value v' of the join attribute B the corresponding bit is set: $B[h(v')] = 1$. The whole process is shown in the following algorithm:

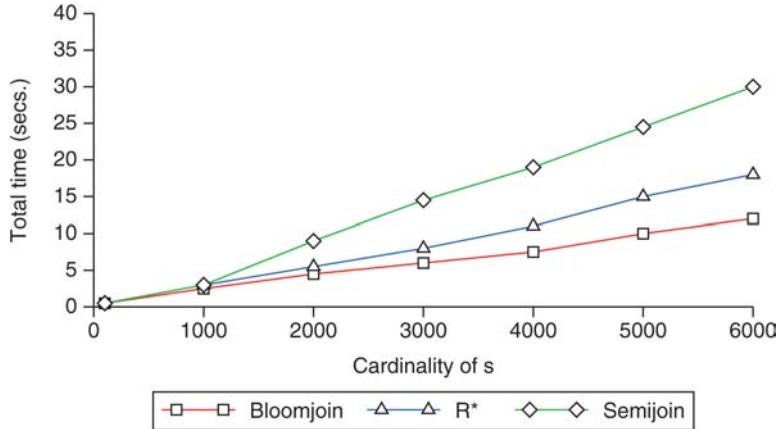
1. At site 1: for each $v \in \pi_A(R)$ set $B[h(v)] = 1$ and send B to site 2
2. At site 2: derive $S' = \{t \in S | B[h(t.C)] = 1\}$
3. Send S' to site 1
4. At site 1: $R \bowtie_{A=C} S'$ is computed producing a result equivalent to $R \bowtie_{A=C} S$

This algorithm is illustrated by an example in Fig. 2 using a simple hash function $h(v) = v \bmod 7$. Applying h to column A of relation R produces the vector of seven bits ([0111000]) which is used to probe the S -tuples at site 2 by computing $h(C)$. Note, that \checkmark indicates a match and $-$ a non-match.

Note that a hash function is usually not injective and therefore the problem of collision occurs, i.e., for different values $v_1 \neq v_2$ one can have $h(v_1) = h(v_2)$. Thus, useless tuples are sent to site 1 in step 3 which will not contribute to the final result, e.g. the tuple with $C = 8$ in this example. This problem can be mitigated by choosing a bit vector of an appropriate length. An alternate



Semijoin. Figure 2. Example of hash filter join.



Semijoin. Figure 3. Comparison of semijoin algorithms.

approach is to use multiple hash functions h_1, \dots, h_k together with the associated bit vectors B_1, \dots, B_k and to set the bits for a value v in each bit vector:

$$B_1[h_1(v)] = 1, B_2[h_2(v)] = 1, \dots, B_k[h_k(v)] = 1$$

All these bit vectors are sent to site 2 and used there for probing. A tuple $t \in S$ qualifies only to be a candidate tuple if all bits are set to 1, i.e., if the result of the bitwise AND is 1. It can be shown that with an increasing k the collision probability comes close to 0.

Key Applications

The main application of semijoin techniques is distributed join processing, where the semijoin acts as a reducer. Though, experimental work has shown that the computational overhead is typically higher than the savings in transfer cost, particularly the hash filter strategy is often an attractive alternative.

Variants of the semijoin are also used for processing queries in heterogeneous databases where a component database provides only limited query capabilities,

e.g. selections with parameters (also called bindings). If a set of tuples is sent to the component database as binding parameter, this corresponds in fact to the semijoin strategy.

Finally, semijoins are also useful for processing star queries in data warehouses. Here, the semijoin technique is exploited for joining each dimension table with the fact table (or more exactly an index on the fact table) in order to collect the rowids of the fact tuples. Then, the intersection of all rowid sets is computed which is finally used to retrieve the tuples from the fact table.

Experimental Results

Mackert and Lohman [7] report results of an experimental analysis of the performance of distributed join strategies in the R* system. Though, the experiments were conducted on a hardware which was up-to-date in the eighties (e.g., a highspeed network with 4 Mbit/s effective transfer rate), the general trend of the results is still valid.

Figure 3 shows the results of a comparison of several strategies for computing $R \bowtie S$ where the cardinality of R was $|R| = 1,000$ and the cardinality of S varied from 100 to 6,000.

In this experiment, the hash filter join clearly outperformed the other strategies. Only for small cardinalities where the inner relation S fits into the buffer, the semijoin has advantages. The third join variant was the R* strategy of shipping one relation to the other site and exploiting local indexes for join processing.

Cross-references

- ▶ [Distributed Join](#)
- ▶ [Evaluation of Relational Operators](#)
- ▶ [Semijoin Program](#)

Recommended Reading

1. Babb E. Implementing a relational database by means of specialized hardware. *ACM Trans. Database Syst.*, 4(1):1–29, 1979.
2. Bernstein P.A. and Chiu D.-M.W. Using semi-joins to solve relational queries. *J. ACM*, 28(1):25–50, 1981.
3. Bernstein P.A., Goodman N., Wong E., Reeve C.L., and Rothnie Jr. Query processing in a system for distributed databases (SDD-1). *ACM Trans. Database Syst.*, 6(4):602–625, 1981.
4. Bloom B.H. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
5. Hevner A.R. and Yao S.B. Query Processing in distributed database systems. *IEEE Trans. Software Eng.*, 5(3):177–182, 1979.
6. Lu H. and Carey M. Some experimental results on distributed join algorithms in a local network. In Proc. 11th Int. Conf. on Very Large Data Bases, 1985, pp. 229–304.

7. Mackert L.F. and Lohman G. R* optimizer validation and performance evaluation for local queries. In Proc. ACM SIGMOD Int. Conf. on Management on Data, 1986, pp. 4–95.
8. Özsu M.T. and Valduriez P. Principles of distributed database systems, 2nd edn. Prentice-Hall, 1999.
9. Valduriez P. Semi-join algorithms for distributed database machines. In J.-J. Schneider (ed.). *Distributed Data Bases*. North-Holland, 1982, pp. 23–37.
10. Valduriez P. and Gardarin G. Join and semi join algorithms for a multiprocessor database machine. *ACM Trans. Database Syst.*, 9(1):133–161, 1984.

Semijoin Filter

- ▶ [Semijoin](#)

Semi-Structured Data

- ▶ [Semi-structured Data Model](#)

Semi-Structured Query Languages

- ▶ [Semi-structured query language](#)

Semijoin Program

STÉPHANE BRESSAN

National University of Singapore, Singapore,
Singapore

Synonyms

- [Semijoin reducer](#)

S

Definition

A semijoin program is a query execution plan for queries to distributed database systems that uses semijoins to reduce the size of relation instances before they are transmitted and further joined. Yet the reduction itself requires that a projection of the relation instances involved in the join onto the join attributes be transmitted. The maximum amount of reduction can be achieved by a semijoin program called a full reducer. Full reducers that do not require the computation of a fixpoint exist for acyclic queries. Fully reducing relation instances is rarely beneficial. However semijoin

programs partially reducing selected relation instances may be an effective optimization when the dominant cost of query execution is communication. Considering semijoin programs considerably increases the distributed query optimization search space.

Historical Background

Semijoin programs were first introduced to improve the performance, input/output operations and communication of database applications running on database machines [7] and on parallel database machines [10]. These machine were possibly equipped with specialized hardware such as filters efficiently implementing semi-joins. SDD-1 [3] is the first distributed database management system making use of semijoin programs for query optimization. Bernstein and Chiu, in [1], review the algorithms for these early applications. In these approaches, reduction of relation instances and the other steps of global and local optimization are conducted as successive phases of the distributed query optimization. Stoker et al. [8], revisit semijoin programs for modern applications and empirically evaluate their usefulness. The authors propose dynamic programming query optimization algorithms that integrate the selection of selected semijoin reducers with join ordering into a single phase.

The hyper-graph representation of relational queries and the notion of acyclic queries is from Fagin [5]. Bernstein et al. introduce comparable notions [2] and define the notion of full reducer. Several authors, see for instance [11] and, more recently [6], have discussed the relationship between reducers and constraint satisfaction problems in the context of database query optimization.

The textbook [4] gives a good overview of semijoin programs and their use in early distributed database systems. It describes the use of semijoin programs in SDD-1 while the presentation in [9] emphasizes the notions of reducers, full reducers, and reduction algorithms.

Foundations

Cyclic and Acyclic Query Hyper-Graphs

For the sake of simplicity, queries are considered which contain natural joins only. The hyper-graph representing such queries is composed of vertices corresponding to attributes and hyper-edges corresponding to relations.

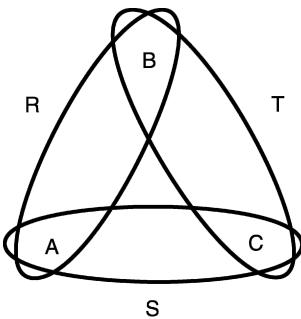
Given, for instance, the three relations $R(A, B)$, $S(A, C)$ and $T(C, B)$, the query $R \bowtie (S \bowtie T)$ is the natural join of R with the natural join of S and T . Its hyper-graph is represented on Fig. 1.

In addition, consider the relation $U(C, D)$. The query $R \bowtie (S \bowtie U)$ is the natural join of R with the natural join of S and U . Its hyper-graph is represented on Fig. 2.

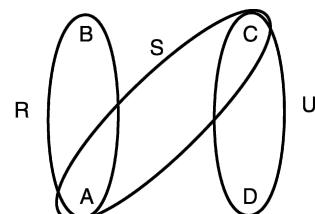
An *ear* of a hyper-graph is a hyper-edge that contains a vertex that does not belong to any other hyper-edge. R and U in Fig. 2. are ears. Notice that removing one ear may create new ears. If all hyper-edges of a hyper-graph can be removed by iteratively removing ears, the hyper-graph is said to be acyclic. The hyper-graph of Fig. 2. is acyclic. The ears U , S and R can be removed in this order, for instance. The hyper-graph of Fig. 1. is cyclic: none of the hyper-edges is an ear.

Reducer and Full Reducer

A reducer for a relation R with respect to a query Q is a program of semijoins with other relations in Q applied to R , such that R can be replaced by the result of the program in Q without changes in the result of the



Semijoin Program. Figure 1. Hyper-graph representing $R \bowtie (S \bowtie T)$.



Semijoin Program. Figure 2. Hyper-graph representing $R \bowtie (S \bowtie U)$.

query Q. In other words, the semijoins possibly remove some tuples that do not contribute to the query Q. The program $R \bowtie S$ is a reducer of R in both queries of Figs. Fig 1. and Fig 2, respectively. The reader can verify that with the instances of R and S given in Fig. 3, the tuple R (1, 2) is removed from R by the semijoin. This tuple neither contributes to the result of the query of Fig. 1 nor to the result of the query of Fig. 2.

A full reducer is a reducer that eliminates all the tuples that do not contribute to the result of the query for any instance of the relations in the query. The program $R \bowtie (S \bowtie U)$ is a full reducer of R for the query of Fig. 2. For the instances of relations R, S and U given in Fig. 3, it reduces R to the instance given in Fig. 4.

Fully reducing R for the query of Fig. 1. requires a fixpoint program that iteratively applies semijoin until no more tuples can be eliminated. It is clear that the number of iterations in the fixpoint depends on the actual instances. For the instances R, S and T in Fig. 3, the query of Fig. 1. denotes an empty result. R can only be reduced to the empty relation by if one iteratively

applies the semijoins $S \bowtie T$, $R \bowtie S$ and $T \bowtie R$ until a fixpoint is reached. In the example the fully reduced instance of R is empty. This is not always the case.

There always exists a full reducer for acyclic queries while cyclic queries always require a fixpoint iteration.

Semijoins, Distributed Query Optimization and Semijoin Programs

Given two relation instances R and S located on two different servers, the computation of the join $R \bowtie S$ would normally consist in sending a copy of either one of the two instances from the server where it resides to the other server. Depending on the attributes of S, on the join condition and on the selectivity of the join, the volume of data transferred might be reduced significantly if the projection of S onto the join attributes is shipped to the server on which R is located (or conversely without loss of generality) where they can be used to compute the semijoin $R \bowtie S$. The result is shipped to the site of S where the original join can be computed.

Example instances of the relations $R(A, B)$ and $S(A, C)$ allocated to server S1 and S2, respectively, are given in Fig. 5.

The natural join of R and S is equivalent to the following expressions.

$$(R \bowtie \pi_A(S)) \bowtie S = (R \bowtie S) \bowtie S$$

The projection $\pi_A(S)$ is executed on S2. The result is sent to S1 and used to compute the semijoin $(S \bowtie R)$. The result of the semijoin for the instances of Fig. 5. is given in Fig. 6. The result of the semijoin is sent to S2. The join $(R \bowtie S) \bowtie S$ is computed on S2.

R	
A	B
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9

S	
A	C
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8

T	
C	B
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8

U	
C	D
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8

Semijoin Program. Figure 3. Instances of R, S, T, and U.

$R \bowtie (S \bowtie U)$	
A	B
3	4
4	5
5	6
6	7
7	8
8	9

Semijoin Program. Figure 4. R fully reduced.

R	
A	B
1	1
1	2
2	1
2	2
3	1
3	2
4	1
4	2

S	
A	C
1	1
1	2
2	3
2	4
5	1
5	2
6	3
6	4

Semijoin Program. Figure 5. The instance of relation R is on site S1 and the instance of relation S is on site S2.

<i>R</i>	\bowtie	<i>S</i>
A		B
1		1
1		2
2		1
2		2

Semijoin Program. Figure 6. Semijoin of *R* and *S*.

This plan is valuable if the volume of data in the projection of *S* on *A* plus the volume of data in the results of its join with *R* is significantly smaller than the volume of data in *S*.

$$\text{volume}(\pi_A(S)) + \text{volume}(S \bowtie (\pi_A(S) \bowtie R)) \text{volume}(S)$$

In the example, if each integer is 4 bytes, by using a semijoin, $4 \times 4 + 4 \times 2 \times 4$ bytes = 48 bytes are transmitted, instead of $8 \times 2 \times 4$ bytes = 64 bytes.

Semijoin programs are implementation of (full or partial) reducers by means of semijoins. Their traditional application to distributed query optimization consists in the reduction of fragments before fragments are shipped from one server to another, as illustrated in the example above. It is rarely beneficial to fully reduce relations, even for acyclic queries.

Considering semijoin programs for query optimization significantly increases the number of candidate query execution plans and, therefore, the search space of the optimization algorithm.

Key Applications

Semijoin programs have been designed and used for the optimization of queries in early distributed database systems, at a time when communication, i.e., data transmission, was the dominant cost. Although semijoin programs fell in desuetude, the authors of [8] argue that they cannot only significantly reduce communication cost in modern distributed database systems, but also allow a better utilization of resources for some applications in both centralized and distributed systems.

Cross-references

- ▶ Bloom Join
- ▶ Distributed Join
- ▶ Distributed Query Processing
- ▶ Semijoin

Recommended Reading

1. Bernstein P.A. and Chiu D.-M. W. Using semi-joins to solve relational queries. *J. ACM*, 28(1):25–40, 1981.
2. Bernstein P.A. and Goodman N. Power of natural semijoins. *SIAM J. Comput.*, 10(4):751–771, 1981.
3. Bernstein P.A., Goodman N., Wong E., Reeve C.L., and Rothnie J.B. Query processing in a system for distributed databases (SDD-1) *ACM Trans. Database Syst.*, 6(4):602–625, 1981.
4. Ceri S. and Pelagatti G. *Distributed databases: Principles and systems*. McGraw-Hill, 1984.
5. Fagin R. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.
6. Lal A. and Choueiry B.Y. Constraint processing techniques for improving join computation: a proof of concept. In Proc. 1st Int. Symp. on Applications of Constraint Databases, 2004, pp. 149–167.
7. Ozkarahan E.A., Schuster S.A., and Sevcik K.C. Performance evaluation of a relational associative processor *ACM Trans. Database Syst.*, 2(2):175–195, 1977.
8. Stocker K., Kossmann D., Braumandl R., and Kemper A. Integrating semi-join-reducers into state of the art query processors. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 575–584.
9. Ullman J.D. *Principles of Database and Knowledge-Base Systems*, Vol. II. Computer Science, 1989.
10. Valduriez P. and Gardarin G. Join and semijoin algorithms for a multiprocessor database machine. *ACM Trans. Database Syst.*, 9(1):133–161, 1984.
11. Wallace M., Bressan S., and Provost T.L. Magic checking: constraint checking for database query optimization. In Proc. ESPRIT WG CONTESSA Workshop on Constraint databases and Applications. LNCS, Vol. 1034. Springer, 1995, pp. 148–166.

Semijoin Reducer

- ▶ Semijoin Program

Semi-Streaming Model

- ▶ Graph Mining on Streams

Semi-Structured Data

SERGE ABITEBOUL

INRIA-Saclay, Ile-de-France, Orsay, Cedex, France

Synonyms

XML (almost)

Definition

A semi-structured data model is based on an organization of data in labeled trees (possibly graphs) and on query languages for accessing and updating data. The labels capture the structural information. Since these models are considered in the context of data exchange, they typically propose some form of data serialization, i.e., a standard representation of data in files. Indeed, the most successful such model, namely XML (that is promoted by the W3C), is often confused with its serialization syntax. XML equipped with query/update language [10] is a semi-structured data model.

Semi-structured data models are meant to represent from very structured to very unstructured information, and in particular, irregular data. In a structured data model such as the relational model [9], one distinguishes between the type of the data (schema in relational terminology) and the data itself (instance in relational terminology). In semi-structured data models, this distinction is blurred. One sometimes speaks of schemaless data although it is more appropriate to speak of self-describing data. Semi-structured data may possibly be typed. For instance, tree automata have been considered for typing XML. However, semi-structured data applications typically use very flexible and tolerant typing or sometimes no typing at all.

Historical Background

Before the Web, publication of electronic data was limited to a few scientific and technical areas. With the Web and HTML, it rapidly became universal. HTML is a format meant for presenting documents to humans. However, a lot of the data published on the Web is produced by machines. Moreover, it is more and more the case that Web data are consumed by machines. Since HTML is not appropriate for machine processing, this lead in the 1990's to the development of *semi-structured data models* and most importantly of a new standard for the Web, namely

XML. The use of a semi-structured data model as a standard for *data representation* and *data exchange* on the Web brought important improvement to the publication and reuse of electronic data by providing a simple syntax for data that is machine-readable and at the same time, human readable (with the help of the so-called “style-sheets”).

Semi-structured data models may be viewed, in some sense, as bringing together two cultures that were for a long while seen as irreconcilable, document systems (with notably SGML [8]) and database systems (with notably relational systems [9]). From a model perspective, there are many similarities with the object database model [5]. Indeed, like XML, the object database model is also based on trees, provides an object API, comes equipped with a query language and offers some form of serialization. A main difference is that the very rigorous typing of object databases was abandoned in semi-structured data models.

The articulation of the notion of semi-structured data may be traced to two simultaneous origins, the OEM model at Stanford [3,6] and the UnQL model at U. Penn [4].

Specific data formats had been previously proposed and even became sometimes popular in specific domains, e.g. ASN.1 [7]. The essential difference between data exchange formats and semi-structured data models is the presence of high level query languages in the latter. A query language for SGML is considered in [2]. Languages for semi-structured data models such as [3,4] then paved the way for languages for XML [10].

Foundations

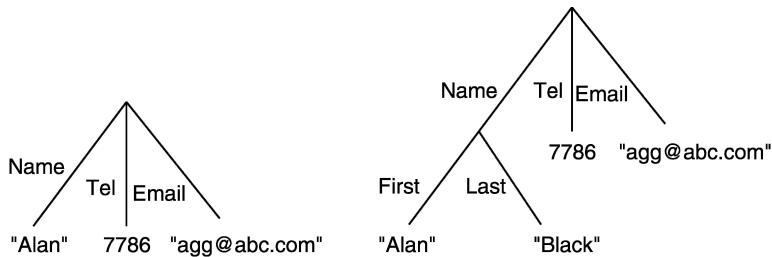
One can start with an idea familiar to Lisp programmers of association lists, which are nothing more than label-value pairs and are used to represent record-like or tuple-like structures:

```
{name: "Alan," tel: 2157786, email:  
"agb@abc.com"}
```

This is simply a set of pairs such as name: "Alan" consisting of a label and a value. The values may themselves be other structures as in:

```
{name: {first: "Alan," last: "Black"},  
tel: 2157786,  
email: "agb@abc.com"}
```

This data may be represented graphically with nodes denoting object, connected by edges to values,



Semi-Structured Data. Figure 1. Tree representation.

see Fig. 1. Departing from the usual assumption made about tuples or association lists that labels are unique, duplicate labels may be allowed as in:

```
{name: "alan," tel: 2157786, tel: 2498762 }
```

The syntax makes it easy to describe sets of tuples as in:

```
{person: {name: "alan," phone: 3127786,
email: "agg@abc.com"},

person: {name: "sara," phone: 2136877,
email: "sara@math.xyz.edu"},

person: {name: "fred," phone: 7786312,
email: "fds@acme.co.uk"}}
```

Furthermore, one of the main strengths of semi-structured data is its ability to accommodate variations in structure, e.g., all the Person tuples do not need to have the same type. The variations typically consist of missing data, duplicated fields or minor changes in representation, as in the following example:

```
{person: {name: "alan," phone: 3127786,
email: "agg@abc.com"},

person: &314
    {name: {first: "Sara," last: "Green"},

phone: 2136877,
email: "sara@math.xyz.edu,"

spouse: &443

person: &443
    {name: "fred," Phone: 7786312, Height:
183,
spouse: &314}}
```

Observe how identifiers (here &443 and &314) and references are used to represent graph data. It should be obvious by now that a wide range of data structures, including those of the relational and object database models, can be described with this format.

As already mentioned, in semi-structured data, the conscious decision is made of possibly not caring

about the type the data might have, and serialize it by annotating each data item explicitly with its description (such as name, phone, etc). Such data are called *self-describing*. The term *serialization* means converting the data into a byte stream that can be easily transmitted and reconstructed at the receiver. Of course self-describing data wastes space, since these descriptions are repeated for each data item, but more interoperability is achieved, which is crucial in the Web context.

There have been different proposals for semi-structured data models. They differ in choices such as labels on nodes versus on edges, trees versus graphs, ordered trees versus unordered trees. Most importantly, they differ in the languages they offer.

Key Applications

The main applications of semi-structured data models are found on the Web.

First, semi-structured data models and XML are very useful for data *publication*. XML is also serving as a universal *data exchange* format in a wide variety of fields, from bioinformatics to e-commerce. It presents the advantage compared to previous formats that it comes equipped with an array of available software such as parsers or programming interfaces. Also, the flexibility of the typing in semi-structured data models turns out to be essential for *data integration*, and in particular in the integration of heterogeneous data in mediator systems.

Cross-references

- ▶ Document representations (incl. native and relational)
- ▶ Semi-Structured Data Model
- ▶ W3C XML Query Language
- ▶ XML
- ▶ XML Types

Recommended Reading

1. Abiteboul S., Buneman P., and Suciu D. Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann, 1999.
2. Abiteboul S., Cluet S., Christophides V., Milo T., Moerkotte G., and Simeon J. Querying documents in object databases. *Int. J. Digit. Libr.*, 1(1):5–19, 1997.
3. Abiteboul S., Quass D., McHugh J., Widom J., and Wiener J. The Lorel query language for semistructured data. *Int. J. Digit. Libr.*, 1(1):68–88, 1997.
4. Buneman P., Davidson S., and Suciu D. Programming constructs for unstructured data. In Proc. 5th Int. Workshop on Database Programming Languages, 1995.
5. Cattell R.G.G. The Object Database Standard: ODMG-93. Morgan Kaufmann Publishers, 1994.
6. Papakonstantinou Y., Garcia-Molina H., and Widom J. Object exchange across heterogeneous information sources. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 251–260.
7. Specification of Abstraction Syntax Notation One (ASN.1), ISO Standard 8824, Information Processing System, 1987.
8. Standard Generalized Markup Language (SGML), ISO 8879, 1986.
9. Ullman J.D. Principles of Database and Knowledge-Base Systems, Vol. I: Classical Database Systems. Computer Science, 1988.
10. XQuery. XQuery 1.0: An XML query language. <http://www.w3.org/TR/Xquery>

Semi-Structured Data Model

DAN SUCIU

University of Washington, Seattle, WA, USA

Synonyms

[Semi-Structured data](#)

Definition

The semi-structured data model is designed as an evolution of the relational data model that allows the representation of data with a flexible structure. Some items may have missing attributes, others may have extra attributes, some items may have two or more occurrences of the same attribute. The type of an attribute is also flexible: it may be an atomic value, or it may be another record or collection. Moreover, collections may be heterogeneous, i.e., they may contain items with different structures. The semi-structured data model is self-describing data model, in which the data values and the schema components co-exist. Formally:

Definition 0.1

A *semi-structured data instance* is a rooted, directed graph in which the edges carry labels representing schema components, and leaf nodes (i.e., nodes without any outgoing edges) are labeled with data values (integers, reals, strings, etc.).

There are two variations of semi-structured data, depending on how one interprets equality. In the object exchange model (OEM) introduced in Tsimmis [8], each node in the data has its own identity, and thus two data instances are “equal” if and only if they are isomorphic, and this corresponds to the bag semantics of collections. In the value-based model introduced in UnQL [2], two data graphs are equal if they are bisimilar; this corresponds to the set semantics of collections.

A variation of the semi-structured data model is one in which labels are placed on nodes rather than edges.

Historical Background

The term *semi-structured data* was introduced by Luniewski et al. in 1993 in a system called Rufus [6,9]. In 1995, Papakonstantinou et al. introduced a data model for semi-structured data (called *object exchange model*, OEM) for a system for integrating heterogeneous databases called Tsimmis [5,8]. In 1995, Buneman et al. introduced a data model for biological data where equality is based on bisimulation [1–3]. The connection between the semi-structured data model and XML was described in 1999 by Deutsch et al., who proposed a query language for XML called XML-QL [4].

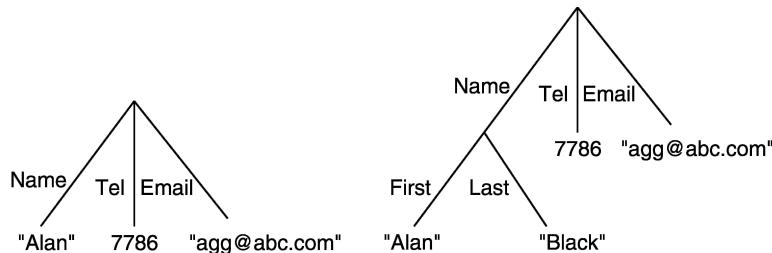
Foundations

Semi-structured data are *schema-less* or *self-describing*. Both the data and its schema is described directly using a simple syntax for sets of label-value pairs, similar to association lists in Lisp. For example:

```
{name: "Alan", tel: 2157786, email: "agb@abc.com"}
```

This is simply a set of pairs such as name: "Alan" consisting of a label and a value. The values may themselves be other structures as in

```
{name: {first: "Alan", last: "Black"}, tel: 2157786, email: "agb@abc.com"}
```



Semi-Structured Data Model. Figure 1. Graph representations of simple structures.

One may represent this data graphically as a node that represents the object, connected by edges to values, see Fig.1.

Unlike in traditional tuples or association lists the labels are not necessarily unique, and duplicate labels are allowed, as in:

```
{name: "alan", tel: 2157786, tel:
2498762}
```

The syntax makes it easy to describe sets of tuples as in

```
{person:
  {name: "alan", phone: 3127786,
   email: "agg@abc.com"} ,
person:
  {name: "sara", phone: 2136877,
   email: "sara@math.xyz.edu"} ,
person:
  {name: "fred", phone: 7786312,
   email: "fds@acme.co.uk"}
}
```

Person tuples do not necessarily have to be of the same type. One of the main strengths of semi-structured data is its ability to accommodate variations in structure. While in principle semi-structured data could become a completely random graph, data instances that are usually found in practice are “close” to some type, and have only minor variations from that type. The variations typically consist of missing data, duplicated fields or minor changes in representation, as in the example below.

```
{person:
  {name: "alan", phone: 3127786,
   email: "agg@abc.com"} ,
person:
  {name: {first: "Sara", last:
  "Green"}, phone: 2136877,
```

```
email: "sara@math.xyz.edu"
},
person:
  {name: "fred", Phone:
  7786312 Height: 183}
}
```

Representing relational databases. It is easy to represent every relational database as a semi-structured data, which happens to have a regular structure. For example the relational database instance:

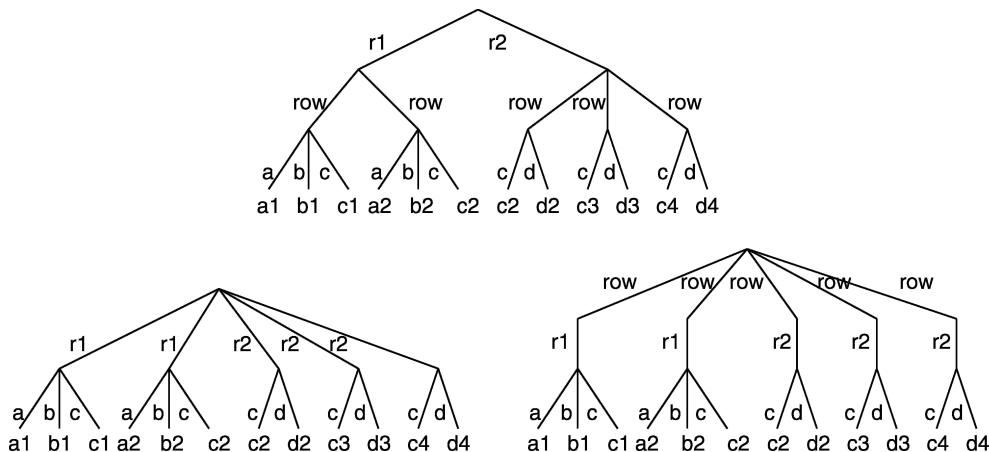
r1	a	b	c
	a1	b1	c1
	a2	b2	c2

r2	c	d
	c2	d2
	c3	d3
	c4	d4

can be described as a set of rows:

```
{r1: {row: {a: a1, b: b1, c: c1},
      row: {a: a2, b: b2, c: c2}}
 },
r2: {row: {c: c2, d: d2},
      row: {c: c3, d: d3},
      row: {c: c4, d: d4}}
 }
```

It is worth noting that this is not the only possible representation of a relational database. Figure 2 shows tree diagrams for the syntax given above and for two other representations of the same relational database.



Semi-Structured Data Model. Figure 2. Three representations of a relational database.

Representing object databases. Consider for example the following collection of three persons, in which Mary has two children, John and Jane. Object identities may be used to construct structures with references to other objects.

```

person: &o1{name: "Mary",
            age: 45,
            child: &o2,
            child: &o3
          },
person: &o2{name: "John",
            age: 17,
            relatives: {mother: &o1,
                        sister: &o3}
          },
person: &o3{name: "Jane",
            country: "Canada",
            mother: &o1
          }
    }
```

The presence of a label such as `&o1` before a structure binds `&o1` to the identity of that structure. This makes it possible to use that label – as a value – to refer to that structure. In this graph representation it is allowed to build graphs with shared substructures and cycles, as shown in Fig.3. The name `&o1`, `&o2`, `&o3` are called *object identities*, or *oid's*. In this figure, arrows are placed on the edges to indicate the direction, which is no longer implicit, like in the tree-like structure.

The *object exchange model* (OEM) was explicitly defined for the purpose of integrating heterogeneous

data sources in Tsimmis. An OEM object is a quadruple $(label, oid, type, value)$, where *label* is a character string, *oid* is the object's identifier, *type* is either *complex*, or some identifier denoting an atomic type (like *integer*, *string*, *gif-image*, etc.). When *type* is *complex*, then the object is called a *complex object*, and *value* is a set (or list) of *oid*'s. Otherwise the object is an *atomic object* and the *value* is an atomic value of that type. Thus OEM data are essentially a graph, but in which labels are attached to nodes rather than edges.

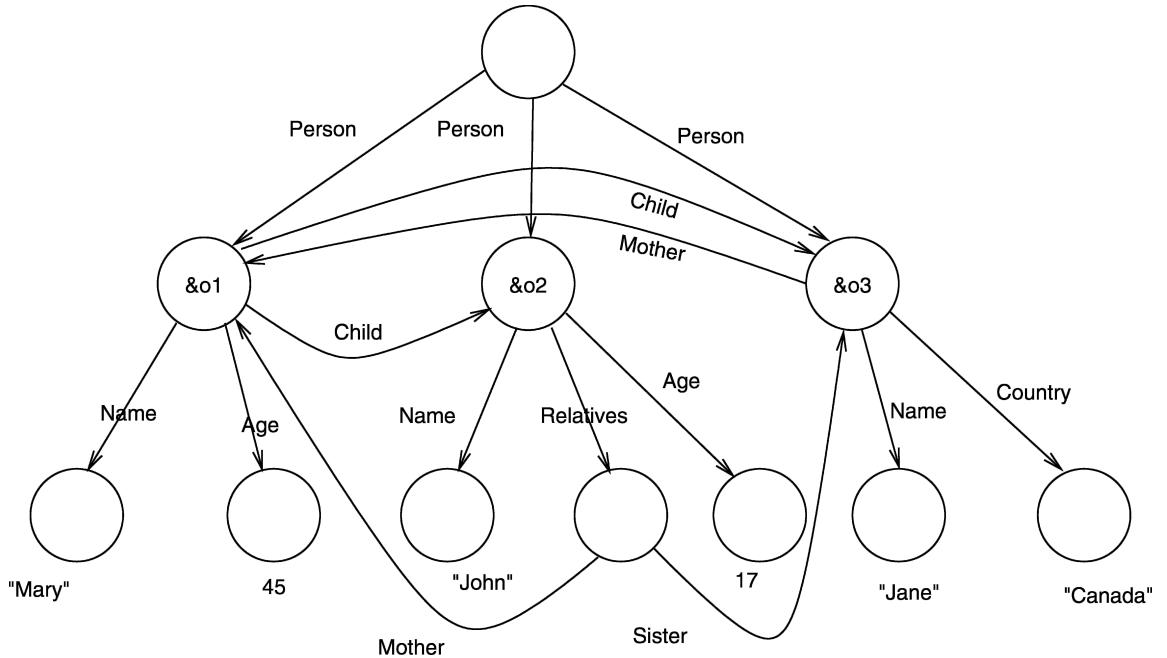
Equality in semi-structured data. A shallow notion of equality simply checks whether two object identifiers are the same, or two data values are equal. Beyond that a deep notion of equality is needed, which addresses the following question: given two semi-structured data instances (i.e., two graphs), do they represent the same data? This question is fundamental in query optimization, since it allows replacement of one query expression with another if the instances they return are equal.

Two notions of deep equality have been considered. One is *graph isomorphism*: two data instances are equal if there exists an isomorphism that preserves the edge labels and the data values.

Definition 0.2

An isomorphism between two semi-structured data instances D_1, D_2 is a function f mapping the nodes of D_1 to the nodes of D_2 such that:

1. f is a bijection.
2. f maps the root of D_1 to the root of D_2 .



Semi-Structured Data Model. Figure 3. A cyclic structure.

3. If there exists an edge from a node x to a node y in D_1 then there exists an edge from the node $f(x)$ to the node $f(y)$ in D_2 and both edges have the same label.
4. Conversely, if there exists an edge from $f(x)$ to $f(y)$ in D_2 then there exists an edge (It follows from the previous condition that the two edges have the same label.) from x to y in D_2 .
5. If a leaf node x in D_1 is labeled with a data value v then the node $f(x)$ in D_2 is labeled with the same data value (It follows from the previous conditions that $f(x)$ is also a leaf node.) v .

In the first interpretation two data instances are equal if there exists an isomorphism between them. For example the two instances $\{a : 3, b : 5, c : 7, b : 9\}$ and $\{b : 5, b : 9, a : 3, c : 7\}$ are equal, because they are isomorphic. On the other hand, the instances $\{a : 3, a : 3, b : 5\}$ and $\{a : 3, b : 5\}$ are not equal. Thus, when restricted to collections this notion of equality corresponds to the bag semantics.

The second notion of equality is based on *bisimulation*: two data instances are equal if there exists a bisimulation:

Definition 0.3

A bisimulation between two semi-structured data instances D_1, D_2 is a relation $R(x, x')$ between the nodes in the two instances s.t.

1. If r_1 is the root node in D_1 and r_2 is the root node in D_2 , then $R(r_1, r_2)$.
2. If $R(x, x')$ holds, and D_1 contains an edge (x, y) with label a , then there exists an edge (x', y') labeled a in D_2 , and $R(y, y')$ holds.
3. Symmetrically, if $R(x, x')$ holds and D_2 contains an edge (x', y') with label a then there exists an edge (x, y) in D_1 labeled a , and $R(y, y')$ holds.
4. If $R(x, x')$ holds and the node x in D_1 is a leaf node, and is labeled with the atomic value v then the node x' in D_2 is also a leaf node and labeled with the same atomic value v (the symmetric property follows automatically).

In the second interpretation, two semi-structured data instances are said to be equal if there exists a bisimulation between them. For example $\{a : 3, a : 3, b : 5\}$ is equal to $\{a : 3, b : 5\}$ because, denoting r, n_1, n_2, n_3 the nodes in the first graph and r', n'_1, n'_2 the nodes in the second graph, the relation $R(r, r')$, $R(n_1, n'_1)$, $R(n_2, n'_1)$, $R(n_3, n'_2)$ is a bisimulation. Thus, the equality based on bisimulation corresponds to the set semantics on collections.

If two data instances are isomorphic, then there always exists a bisimulation between them; the converse does not always hold. Checking whether two data instances are isomorphic is a computationally hard problem. By contrast, checking if two data instances are bisimilar can be done efficiently [7].

Edge vs. node labeled graphs. The model described here is that of an *edge-labeled graph*. A minor variation is one in which nodes are labeled, and this has gained a lot of popularity since the introduction of XML.

Key Applications

The initial motivation for the introduction of semi-structured data was to support the integration of heterogeneous data, and to model non-standard data formats, especially in the bioinformatics domain, s.a. ACEDB and ASN.1. After the introduction of XML, this became the main application of semi-structured data.

Cross-references

- ▶ [Semi-Structured Data](#)
- ▶ [Semi-Structured Query Languages](#)
- ▶ [XML](#)

Recommended Reading

1. Buneman P., Davidson S., and Suciu D. Programming constructs for unstructured data. In Proc. Workshop on Database Programming Languages, 1995.
2. Buneman P., Davidson S., Hillebrand G., and Suciu D. A query language and optimization techniques for unstructured data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 505–516.
3. Buneman P., Fernandez M., and Suciu D. UNQL: A query language and algebra for semistructured data based on structural recursion. VLDB J., 9(1):76–110, 2000.
4. Deutsch A., Fernandez M., Florescu D., Levy A., and Suciu D. A query language for XML. In Proc. 8th Int. World Wide Web Conference, 1999, pp. 77–91.
5. Garcia-Molina H., Papakonstantinou Y., Quass D., Rajaraman A., Sagiv Y., Ullman J., and Widom J. The TSIMMIS project: integration of heterogeneous information sources. J. Intell. Inf. Syst., 8(2):117–132, March 1997.
6. Luniewski A., Schwarz P., Shoens K., Stamos J., and Thomas J. Information organization using Rufus. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 560–561.
7. Paige R. and Tarjan R. Three partition refinement algorithms. SIAM J. Comput., 16:973–988, 1987.
8. Papakonstantinou Y., Garcia-Molina H., and Widom J. Object exchange across heterogeneous information sources. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 251–260.
9. Shoens K., Luniewski A., Schwarz P., Stamos J., and Thomas II J. The Rufus system: Information organization for semi-structured data. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 97–107.

Semi-Structured Database Design

GILLIAN DOBBIE¹, TOK WANG LING²

¹University of Auckland, Auckland, New Zealand

²National University of Singapore, Singapore, Singapore

Synonyms

[XML Database Design](#)

Definition

From a requirements document, a database designer distills the real world constraints and designs a database schema. While the design process for structured data is well defined, the design process for semi-structured data is not as well understood. What is a “good” design for semi-structured databases that captures real world constraints, prevents data redundancy and update anomalies, and allows typical queries to execute quickly?

Historical Background

There was a lot of research into the design of relational databases in the 1970s, and it was found that the design of relational databases involves a trade off between the speed of execution of queries and the updating anomalies caused by maintaining redundant data when updates occur. During logical schema design normalization algorithms are used to reduce redundancy, and during physical design to improve performance some redundancy may be reintroduced, views can be created over the schema, and indexes may be introduced.

Semi-structured data differs from relational data in a number of ways: it is hierarchical, the queries that are posed are more complex, a database may or may not have a schema, and there is no generally agreed upon mathematical foundation. Because of these differences, there is a need for a different design process for semi-structured databases.

Foundations

Consider the XML document in Fig. 1a. It models the courses that students are taking within a department, and the grade of each student taking a course. What is the best way to organise this information? There are various possibilities, such as modeling *course* as a subelement of *student*, or modeling *student* as a subelement of *course*. While both of these options

Semi-Structured Database

- ▶ [Graph Database](#)

```

<department>
  <name>Computer Science</name>
  <student>
    <stuNo>123456</stuNo>
    <stuName>Bob Smith</stuName>
    <course>
      <code>CS101</code>
      <title>Introduction to Computer Science</title>
      <grade>A</grade>
    </course>
    <course>
      <code>CS105</code>
      <title>Data Structures</title>
      <grade>B</grade>
    </course>
  </student>
  <student>
    <stuNo>234567</stuNo>
    <stuName>Mary Brown</stuName>
    <course>
      <code>CS101</code>
      <title>Introduction to Computer Science</title>
      <grade>B</grade>
    </course>
  </student>
</department>

```

a An XML Document

```

<department>
  <name>Computer Science</name>
  <student>
    <stuNo>123456</stuNo>
    <stuName>Bob Smith</stuName>
    <cGrade>
      <code>CS101</code>
      <grade>A</grade>
    </cGrade>
    <cGrade>
      <code>CS105</code>
      <grade>B</grade>
    </cGrade>
  </student>
  <student>
    <stuNo>234567</stuNo>
    <stuName>Mary Brown</stuName>
    <cGrade>
      <code>CS101</code>
      <grade>B</grade>
    </cGrade>
  </student>
  <course>
    <code>CS101</code>
    <title>Introduction to Computer Science</title>
  </course>
  <course>
    <code>CS105</code>
    <title>Data Structures</title>
  </course>
</department>

```

b A normalized XML Document

Semi-Structured Database Design. Figure 1. An original and normalized XML Document.

seem quite natural, Fig. 1a demonstrates that these options involve repeating information. The details of the course, i.e., title in this example, are repeated for each student that takes that course. Consequently, if the title of a course changes, it must be updated in every *title* element of the course. Brandin [2] describes a simple way to define XML documents, in which the modeler designs simple forms entailing the information to be modeled, and uses the form headings as tags and the entries on the form as data. While the simplicity of methods like this is appealing, they can lead to similar anomalies that arise with relational databases.

In order to ensure a “good” design for semi-structured databases, the following steps must be followed:

1. Choose a data model that is able to represent the semantics necessary for modeling semi-structured data.
2. Capture the semantics of the data that will be stored, either by:

- (a) Extracting the schema from a set of documents and discovering the semantics in a data model, or
- (b) Studying the constraints in the real world and capturing them in a data model.
3. Reorganize the schema into a normalized schema to avoid replication of data in the XML documents.
4. Consider the typical query set and reorganize the schema to improve the performance of typical queries, perhaps by introducing controlled replication of data.
5. Consider the users of the system and define views over the data for individual users or groups of users.

A data model for semi-structured database design needs to model the logical structure of the schema from a real world perspective, much like an ER diagram [4], while also modeling the physical aspects that are representative of XML documents, such as the hierarchical relationship between elements. This enables a designer to first model the real world constraints on the data, and using the same diagram, capture the extra constraints that are

introduced with XML. For example, consider again the scenario captured in Fig. 1a. From a real world perspective the data model must model the relationship between students and courses, and from an XML perspective the data model must be able to capture whether student should be modeled as a subelement of course or vice-versa. The data model must also capture the participation constraints between elements, because they also change the way the data are modeled. For example, there is a many-to-many relationship between courses and students and this will be modeled differently than a one-to-many relationship say between department and employees. In order to capture these requirements, the data model must be able to model n-ary relationship sets, cardinality, participation and uniqueness constraints, ordering, irregular and heterogeneous structures, for both data- and document-centric data. One such data model is ORA-SS (Object Relationship Attribute Data Model for Semi-Structured Data) [6].

One of the advantages of semi-structured data is that it is self-describing, and so it does not require a schema. However, after XML was introduced, it was soon realized that schemas offer many benefits, and as a consequence schema languages have been defined for XML recently [3,8]. Some semantics of a document can be extracted from the schema if one exists or extracted from the document. In [7,9] and Chap. 4 of [6], the authors have described how an approximate schema

can be extracted from semi-structured data. The algorithms generally follow two steps. The first step extracts the structural information, such as elements and their subelements. The second step infers semantics from the original document, such as key attributes, attributes of object classes vs attributes of relationship types, and participation constraints. Normalization is used to identify and reduce anomalies, and a key ingredient in normalization algorithms is functional dependencies. Consider the real world constraints that should be modeled in the document in Fig. 1a. There is a many-to-many relationship between elements *student* and *course*, attributes *stuNo* and *stuName* belong to *student*, attributes *code* and *title* belong to *course*, and attribute *grade* belongs to the relationship type between *student* and *course*. These relationships may be modeled as functional dependencies, such as *stuNo* → *stuName*, *code* → *title*, and {*stuNo*,*code*} → *grade*. Yu and Jagadish [10] describe a system, DiscoverXFD, which given an XML document, discovers XML functional dependencies and data redundancies. However, multi-valued dependencies (MVDs) are also very important in relational database design, such as in the definition of 4NF. Like with relational databases, functional dependencies cannot describe all redundancy in XML databases, so MVDs must also be considered.

The motivation for normalization algorithms for semi-structured databases is similar to that for relational databases, namely the identification of

```

<department>
  <name>Computer Science</name>
  <student>
    <stuNo>123456</stuNo>
    <stuName>Bob Smith</stuName>
    <C_G>
      <course>CS101</course>
      <grade>A</grade>
    </C_G>
    <C_G>
      <course>CS105</course>
      <grade>B</grade>
    </C_G>
  </student>
  <student>
    <stuNo>234567</stuNo>
    <stuName>Mary Brown</stuName>
    <C_G>
      <course>CS101</course>
      <grade>B</grade>
    </C_G>
  </student>
<course>
  <code>CS101</code>
  <title>Introduction to Computer Science</title>
  <S_G>
    <student>123456</student>
    <grade>A</grade>
  </S_G>
  <S_G>
    <student>234567</student>
    <grade>B</grade>
  </S_G>
</course>
<course>
  <code>CS105</code>
  <title>Data Structures</title>
  <S_G>
    <student>123456</student>
    <grade>B</grade>
  </S_G>
</course>
</department>
```

Semi-Structured Database Design. Figure 2. A symmetric relationship in an XML Document for *department*.

redundant data that lead to update anomalies. The normalization algorithms must recognize the hierarchical structure of semi-structured data, the participation constraints of both parents and children in hierarchical relationships, and whether an attribute is an attribute of an object class or the attribute of a relationship type. The algorithm described in [1] converts an arbitrary DTD into a well-designed one, using path functional dependencies. They show that the normal form that they define, XNF, generalizes BCNF for XML documents. This algorithm works with the semantics available in the DTD along with additional functional dependencies, however it does not consider MVDs, so documents in XNF may still contain redundancy. Using the semantics expressed in ORA-SS, it is possible to capture the multi-valued attributes of object classes and relationship types, avoiding the redundancy because of the existence of multi-valued attributes or MVDs. Note each multi-valued attribute of an object class is a MVD, e.g. if a student can have many hobbies, $studNo \rightarrow hobby$. As hobby is not involved in any functional dependencies, XNF will not be able to detect such redundancy but the normalization algorithm defined in [6] deals with this case.

```
<department>
  <name>Computer Science</name>
  <course>
    <code>CS101</code>
    <title>Introduction to Computer Science</title>
    <grade>A</grade>
  </course>
  <course>
    <code>CS105</code>
    <title>Data Structures</title>
    <grade>B</grade>
  </course>
  <course>
    <code>CS101</code>
    <title>Introduction to Computer Science</title>
    <grade>B</grade>
  </course>
</department>
```

a An invalid view over the XML Document in Figure 1a

Figure 1b shows a normalized version of the XML document shown in Fig. 1a. Note that the details of each course are no longer stored for each student that takes the course. These details are extracted and stored in a *course* element which is a sibling to the *student* elements. A new element, *cGrade*, is introduced to record the grade a student scored in a course. The element *code* in *cGrade* is effectively a reference to *course*.

During physical database design, the database schema is tuned to ensure that queries, which are likely to be asked often, will run faster. This can be done by reintroducing redundant data in a controlled way, considering how indexes will improve execution times, or by introducing views. There has not been a lot of work carried out in the area of physical database design or database tuning for semi-structured databases. When reintroducing redundant data, it is necessary to consider both the cost of storage and the cost of updating the redundant data. Consider for example, the XML document in Fig. 1a. The information that is repeated is the title of the course, and it is repeated for every student taking the course. In this case, the title of a course is not likely to be

```
<department>
  <name>Computer Science</name>
  <course>
    <code>CS101</code>
    <title>Introduction to Computer Science</title>
  </course>
  <student>
    <stuNo>123456</stuNo>
    <stuName>Bob Smith</stuName>
    <grade>A</grade>
  </student>
  <student>
    <stuNo>234567</stuNo>
    <stuName>Mary Brown</stuName>
    <grade>B</grade>
  </student>
  <course>
    <code>CS105</code>
    <title>Data Structures</title>
  </course>
  <student>
    <stuNo>123456</stuNo>
    <stuName>Bob Smith</stuName>
    <grade>B</grade>
  </student>
</department>
```

b An valid view over the XML Document in Figure 1a

updated very often. If users were expected to ask a query often that lists the students name and the titles of all the courses a student is taking, then with this profile the designer would opt to retain this level of redundancy. In fact, *title* is an example of a relatively stable attribute, that is an attribute that is updated infrequently. For relatively stable attributes or relatively stable relationship types [5] one needs to consider only the cost of extra storage since these data are not updated often. For all replications ensure that the necessary controls are put in place to maintain the integrity of the data. For example, for relatively stable attributes and relationship types, it is necessary to enforce that replicated data are consistent when the data are inserted.

In some instances, it is not obvious whether to embed one object class inside another or vice versa. Consider the relationship in Fig. 1a. If it is equally likely that users will ask for all the students by course, and all the courses by student then the best design may be a symmetric relationship type. For the pairing required for symmetric relationship types, a reference is duplicated. The consistency of this duplication must be enforced when information is inserted, deleted or updated. A symmetric relationship type is modeled in Fig. 2. In this case, within the *student* element a reference to *course* is stored and in the *course* element a reference to *student* is stored. The *grade* is stored in both *student* and *course*. This approach is similar to a bidirectional relationship with physical pairing in IBM's Information Management System (IMS).

When views are defined, one of the challenges is ensuring that the view over the underlying data is valid. None of the systems available today check the validity of a view against the semantics of the underlying data. In [6], a set of operators is described that can be used to define views, along with guidelines of how the operators can be used to produce valid views. For example, a view over the XML document in Fig. 1a is shown in Fig. 3a. This view is very simple, and was created by extracting all the course elements from the original document. However, this view is invalid because although it lists course codes with their titles, each *course* element also has a grade attached to it. Having removed the context of student, the information about who the grade was assigned to is lost. The operators that are needed to create views are select, drop, join, and swap. The select operator filters

elements for which a particular condition is true and does not alter the schema. For example, a view over the XML document in Fig. 1a can be created using the select operator of all students and courses where the student achieved an "A" grade. The drop operator drops elements or attributes from the source schema. For example, it is possible to create a view by dropping the student elements from the XML document in Fig. 1a. However, it is important that if an element which represents an object class is dropped, then all its attributes or attributes of relationship types in which it participates must also be dropped, otherwise it is possible to create an invalid view such as that shown in Fig. 3a. Note that *grade* is an attribute of the relationship type between the object classes *student* and *course* and should be removed when *student* is dropped. The join operator joins two elements where one contains a foreign key of another. For example, if the underlying schema is that shown in Fig. 2 it would be possible to create a view that joins the course and student elements in Fig. 2 to improve the performance of queries that ask for the courses taken by a student with a specific *stuNo*. In order to guarantee the validity of the drop operator it is necessary to ensure that all ancestors and descendants of the object class being joined are dealt with appropriately. The swap operator exchanges the position of a parent element with a child element. For example, the XML document in Fig. 1a embeds *course* elements in *student* elements. In order to answer the query that asks for the students taking a particular course, it is possible to use the swap operator to create a view where *student* elements are embedded in *course* elements. When the swap operator is used, the relationship types must be updated, and the attributes that belong to the relationship types must also be moved to the appropriate place. The relationship types that need to be updated are those among the elements involved in the swap. Figure 3b shows a valid view over the XML document in Fig. 1a, created by swapping the object classes *student* and *course*.

S

Key Applications

The motivation for studying this area is to ensure that databases that are designed to store semi-structured data will always contain consistent data, and will always provide answers to queries in an acceptable time. The people that will benefit most from this research are the users of semi-structured database systems.

Future Directions

There are a number of key areas that deserve further investigation:

- Definition of a standardized data model for semi-structured data
- Investigation and comparison of the normalization algorithms that have been proposed for semi-structured data
- Investigation and experimentation of physical database design techniques for semi-structured data
- Case studies that show how well the research works in practice
- Cost models for queries over XML databases

Cross-references

- ▶ Entity Relationship Model
- ▶ Normal Form ORA-SS Schema Diagrams.
- ▶ Object Relationship Attribute Model for semi-structured Data
- ▶ Database Design
- ▶ Semi-structured Data Models
- ▶ XML
- ▶ Semi-structured Query Languages

Recommended Reading

1. Arenas M. and Libkin L. A normal form for XML documents. *ACM Trans. Database Syst.*, 29(1):195–232, 2004.
2. Brandin C. Information Modeling with XML. In A. Chaudhri, A. Raschid, and R. Zicari (ed.). *XML Data Management*. Addison-Wesley, 2003, pp. 3–17.
3. Bray T., Paoli J., and Sperberg-McQueen C.M. Extensible markup language (XML) 1.0., 2nd edn. October 2000.
4. Chen P.P. The entity-relationship model – toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36, 1976.
5. Ling T.W., Goh C.H., Lee M.-L. Extending classical functional dependencies for physical database design. *Inf. & Software Tech.*, 38(9):601–608, 1996.
6. Ling T.W., Lee M.L., and Dobbie G. Semistructured Database Design. Springer, 2005.
7. Nestorov S., Abiteboul S., and Motwani R. Extracting schema from semistructured data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 295–306.
8. Thompson H.S., Beech D., Maloney M., and N. Mendelson (eds.). *XML Schema Part 1: Structures*. May 2001. <http://www.w3.org/TR/xmlschema-1>.
9. Wang Q.Y., Yu J.X., and Wong K.F. Approximate graph schema extraction for semi-structured data. In Advances in Database Technology, Proc. 7th Int. Conf. on Extending Database Technology, 2000, pp. 302–317.
10. Yu C. and Jagadish H.V. Efficient discovery of XML data redundancies. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 103–114.

Semi-Structured Query Languages

DAN SUCIU

University of Washington, Seattle, WA, USA

Synonyms

[Semi-Structured query languages](#)

Definition

A query language for semi-structured data allows a user to extract items from a semi-structured data instance, or to transform it into another semi-structured data instance. The first type of expressions are called *queries*, the latter kind of expressions are interchangeably called *queries* or *transformations*. Query languages can be classified along several dimensions:

1. *Expressive power*. What kind of queries or transformations can they express?
2. *Querying vs. restructuring*. Some query languages allow only the extraction of items from the data, others allow the data to be transformed.
3. *Compositionality*. Can the output of a query be used as input in another query expressed in the language, and is the composed transformation still expressible in the same language? Query languages that are restricted to extraction only are not compositional, because the type of their outputs are not semi-structured data instances. Transformation languages may fail to be compositional if the composition of two queries cannot be expressed in the same language.

Historical Background

Query languages for semi-structured data were introduced almost as early as the data model itself. Lorel [1] was one of the first query language for semi-structured data, and it introduced the concept of regular path expressions for navigating paths with partially known structure. The language UnQL [2] emphasized transformations over queries, and introduced structural recursion as a central paradigm for transforming semi-structured data. MSL [5] introduced Skolem functions in a logic-based language for transforming semi-structured data: these were later used extensively in Strudel [4], a system for declarative specification of Web sites. The first language to apply the principles

of semi-structured data query languages to XML was XML-QL [3].

Foundations

Path expressions. A node sequence of edge labels l_1, l_2, \dots, l_n is called a *path expression*. In its most general form the path expression denotes all pairs of nodes (x, y) such that there exist edges $(x, l_1, v_1), (v_1, l_2, v_2), \dots, (v_{n-1}, l_n, y)$. When used in a query language, x is either specified explicitly or assumed by default to be the root node. Thus, a path expression denotes a *set of nodes*, namely the set of nodes y , given the fixed x .

A *regular path expression* is

$e ::= l \mid \epsilon \mid - \mid e^{\cdot} \cdot e \mid ('e') \mid e^{\epsilon} \mid e^{\epsilon *} \mid e^{\epsilon +} \mid e^?$

where l ranges over label, e over expressions and ϵ is the empty expression. The expression e^* stands for 0 or more repeats of e , i.e., for

$\epsilon \mid e \mid e.e \mid e.e.e \mid e.e.e.e \mid \dots$

Also, e^+ stands for one or more repeats, and $e^?$ for 0 or one occurrence of e .

(Regular) path expression form a simple query language for semi-structured data, or can be embedded in a simple query syntax to define such a language. For example the query below returns all nodes X accessible by the path `biblio.book.author`.

```
% Query q1
select X
from biblio.book.author X
```

Note that a query language consisting of path expressions is not compositional, since queries return sets of nodes, not pieces of semi-structured data.

Patterns. A convenient way to combine multiple path expressions is through patterns. Consider the following query, returning all titles that were published before 1979:

```
% Query q2:
select X
from biblio.paper T, T.title X, T.year Y
where Y < 1979
```

Note that here the expression `T.title` starts from the node `T` rather than from the root. The query can be written more concisely as:

```
% Query q3:
select X
```

```
from {biblio: {paper: {title: X, year:
Y}}}
where Y < 1979
```

Here `biblio: paper: title: X, year:Y` is a *pattern*.

Constructors. To return semi-structured data rather than nodes a query language needs to have *constructors*. For example the expressions `authorResult:X` and `result:{title X, author Z}` are constructors below:

```
% Query q4:
select authorResult:X
from biblio.book.author X
```

```
% Query q5:
select result:{title: X, author: Z}
from biblio.paper T, T.title X, T.year
Y, T.author Z
where Y < 1979
```

More complex constructors can built by nesting subqueries inside constructors, as in:

```
% Query q6:
select row:( select author: Y
from X.author Y)
from biblio.book X
```

```
% Query q7:
select row: ( select author: Y, title: T
from X.author Y
X.title T)
from biblio.paper X
where "Roux" in X.author
```

Declarative semantics. The formal semantics of a query is given in three steps. In the first step a set of bindings to all variables is computed, which results in a relation that has one column for every variable in the query, and one row for every binding of these variables. In the second step the predicates in the where clause are applied to select a subset of the rows. In the third step, the constructor is applied to each remaining row. For example in query q6 above, the effect of the first step is to construct a relation of tuples of the form (t, x, y, z) , where t, x, y, z are nodes or atomic values in the input databases. The second step selects only those tuples for which $z < 1979$. The third step constructs a partial graph for each remaining tuple (t, x, y, z) consisting of a root (common among all these graphs), an

edge labeled `result`, and from there two more edges labeled `title` and `author`.

Skolem functions. A Skolem function takes as input some arguments and constructs as output a fresh new node. The essential property of a Skolem function is that if called again at a later time, on the same arguments, then it returns the same node for those argument, not a new one. This allows for duplicate elimination, grouping, and the construction of cyclic outputs. The query below is a standard grouping query that groups publications by their years:

```
% Query q8:
select resultYear f(Y) : {paper:
  {title: X, author: Z}
from biblio.paper T, T.title X, T.year
  Y, T.author Z
```

Here $f(Y)$ is the Skolem function. Without it, the query would construct a separate `resultYear` node for every binding of the variables `T`, `X`, `Y`, and `Z`. The Skolem function determines the query to construct only one node for every year, thus performing a duplicate elimination on years.

The combination of Skolem functions and regular path expressions leads to transformation languages that are not compositional. Consider a semi-structured data instance that represents a binary table R :

```
R: { row: {a: 4, b:8},
      row: {a: 3, b:4},
      row: {a: 3, b:9},
      ...
    }
```

It is known that the relational calculus cannot express the transitive closure of R . The same holds for the language described so far, since its only addition to the relational calculus consists of regular path expressions, but on this simple data instance these expressions can only be applied to very short paths, namely `R.row.a` and `R.row.b`, and therefore do not give extra power. However, the transitive closure can be expressed by first transforming the relation R into a graph, i.e., making all atomic values into nodes, then using a regular expression on this graph to compute the transitive closure. The two queries are

```
% Query q9:
select node f(X) : {value: X, next: f(Y)}
from row T, T.a X, T.b Y
```

```
% Query q10:
select result: {a: X, b:Y}
from node U, U.value X, U.next* V, V.
value Y
```

The first query constructs for every value x of the `a` attribute a new node $f(x)$ with two outgoing edges: one to a leaf holding the value x , and the other to the node $f(y)$. Thus, the edges from $f(x)$ to $f(y)$ in the output graph materialize the implicit graph given by the binary relation R . The second query uses the regular expression `next*` to compute the transitive closure on this graph.

Key Applications

See applications for semi-structured data.

Cross-references

- ▶ [Indexing Semi-Structured Data](#)
- ▶ [Semi-Structured Data](#)
- ▶ [Semi-structured Data](#)
- ▶ [Semi-structured Data Model](#)
- ▶ [Top-k XML Query Processing](#)
- ▶ [XML](#)
- ▶ [XML Tree Pattern, XML Twig Query](#)
- ▶ [XPath/XQuery](#)

Recommended Reading

1. Abiteboul S., Quass D., McHugh J., Widom J., and Wiener J. The Lorel query language for semistructured data, 1996. <http://www-db.stanford.edu/lore/>.
2. Buneman P., Davidson S., and Suciu D. Programming constructs for unstructured data. In Proc. Workshop on Database Programming Languages, 1995.
3. Deutsch A., Fernandez M., Florescu D., Levy A., and Suciu D. A query language for XML. In Proc. 8th Int. World Wide Web Conference, 1999, pp. 77–91.
4. Fernandez M., Florescu D., Kang J., Levy A., and Suciu D. Catching the boat with Strudel: experience with a Web-site management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 414–425.
5. Papakonstantinou Y., Abiteboul S., and Garcia-Molina H. Object fusion in mediator systems. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 413–424.

Semi-Structured Text Retrieval

- ▶ [Biomedical Scientific Textual Data Types and Processing](#)

Semi-Supervised Classification

- ▶ Semi-Supervised Learning

Semi-Supervised Clustering

- ▶ Clustering with Constraints

Semi-Supervised Learning

SUGATO BASU
Google Inc, Mountain View, CA, USA

Synonyms

Semi-supervised classification

Definition

In machine learning and data mining, supervised algorithms (e.g., classification) typically learn a model for predicting an output variable (e.g., class label for classification) from some supervised training data (e.g., data instances annotated with both features and class labels). These algorithms use various techniques of increasing the accuracy of predicting the training data labels, by minimizing a loss function that measures the prediction error on the training data. They also use different regularization methods to ensure that the model does not overtrain on the training data, thereby having good prediction performance on unseen test data.

In semi-supervised learning, unlabeled data (i.e., data instances with only features) are used along with the labeled training data, in an effort to improve the accuracy of the models on the training data as well as provide better generalization performance on unseen data. This paradigm is very useful in practical applications, where unlabeled data are generally available in more abundance (since in most cases unlabeled data are easier to collect) and at a substantially lower cost (e.g., less human effort) than labeled data.

In the published literature, the term semi-supervised learning usually refers to semi-supervised classification. However it can also refer to semi-supervised regression, where unlabeled data are used to improve the performance of a regression algorithm that predicts a real valued output instead of a class

label. A related area of research is semi-supervised clustering, where small amounts of supervision is used to improve the performance of clustering algorithms on unlabeled data.

Historical Background

As outlined in [3] (see Chap. 1 for a detailed history of this area, with more references), the earliest known work [10] that suggested using unlabeled data to improve classification performance is self-learning or self-training. The method first trains a classifier on just the labeled data. In each successive step, the trained model predicts the labels on the unlabeled data instances and uses a part of that (e.g., the labels on which it has highest prediction confidence) as additional labeled data for training in the following step.

Transductive learning [12] is an idea closely related to semi-supervised learning – it uses only the features of the test data instances as unlabeled data to improve the performance of the classifier learned on the labeled data. The main difference between transductive learning and semi-supervised learning is that the latter is an inductive method, i.e., it can generalize to give predictions on new data, while transduction can only give predictions for the given finite set of test data. The relative benefits of these two approaches are well compared in Chap. 25 of [3].

In the 1970s, researchers started to investigate the use of unlabeled data in different classification algorithms, e.g., Fisher's linear discriminant [6], using iterative refinement algorithms like expectation maximization (EM) [5]. The 1990s saw a significant increase of research in semi-supervised learning, especially related to different application areas in text and natural language processing [2,4,7,8,13]. This was accompanied by theoretical analysis of semi-supervised learning algorithms, notably using the PAC learning framework [9].

For detailed historical background and overview of recent work, readers are referred to the excellent surveys [11,14] and book [3] on semi-supervised learning.

Foundations

To concretize the concepts introduced in the definition, consider that the features of the i th data instance in the training data X is represented by the vector x_i , and the corresponding output variable is $y_i \in Y$. In classification, Y is a set of labels, while in regression it a set of real-values. In semi-supervised learning, the

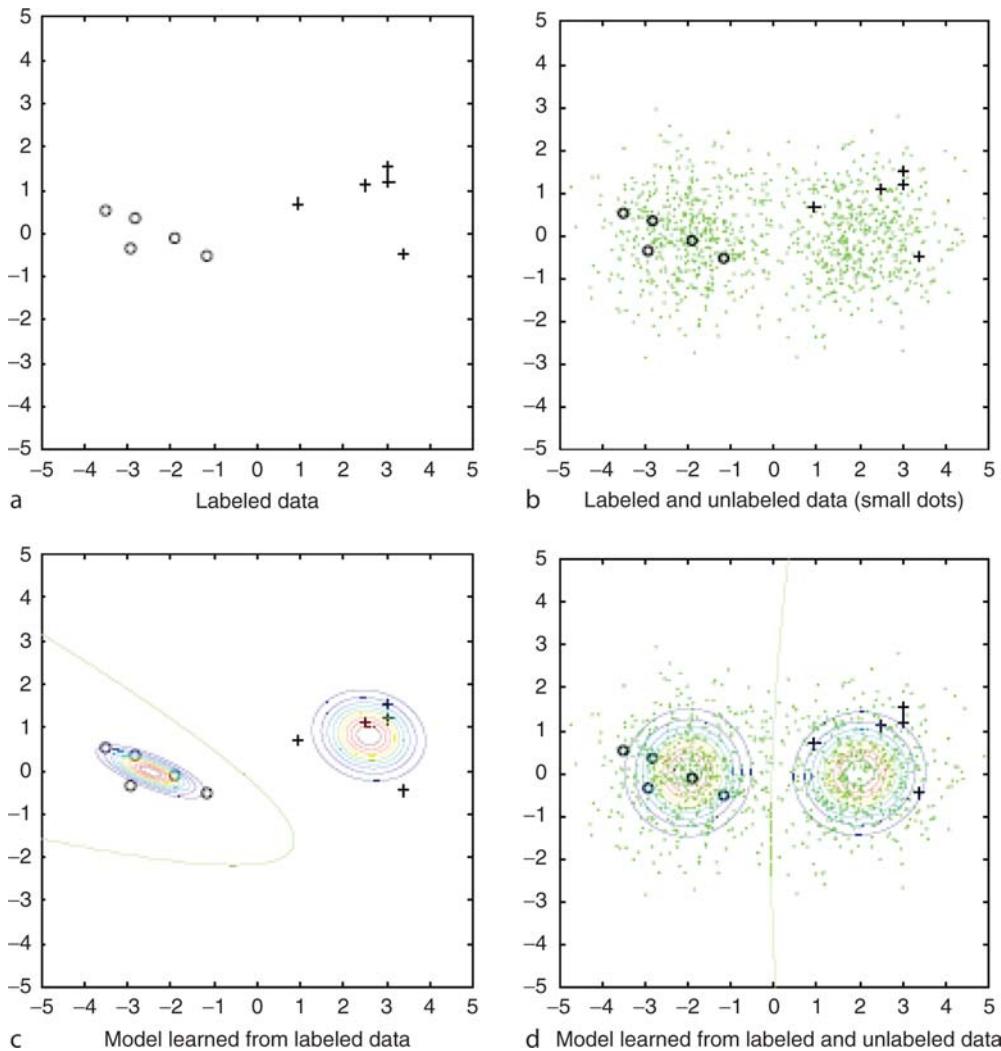
training set X consists of two components: (i) the supervised data X_b where the data instances x_l are annotated with the corresponding y values; (ii) the unsupervised data X_u , where the data instances x_u do not have corresponding y values.

There are different types of semi-supervised learning methods, some of which are listed below:

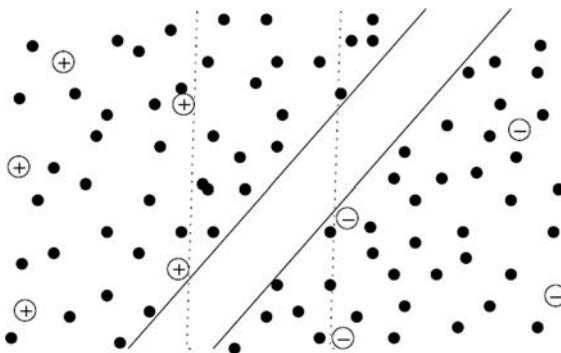
1. *Generative models.* They estimate the class-conditional probability of the data $p(x|y)$, and use the unlabeled data X_u as additional information to improve the model fitting (as shown in Fig. 1). For example, [8] first trains a naive-Bayes classifier C_l on the labeled data X_l and then uses EM to re-train the

classifier on the unlabeled data X_u , using their estimated class probabilities $p(y_u|x_u)$ of the unlabeled data instances (estimated by C_l) as fractional class labels during the EM iterations.

2. *Low density separation.* Models following this principle push the decision boundary of the classifier away from the high-density regions of the unlabeled data. For example, [7] trains a transductive support vector machine (SVM) that maximizes the margin of the SVM classifier on both labeled and unlabeled data – it first trains a classifier C_l using only the labeled data X_b and then uses the predictions $p(y_u|x_u)$ of C_l on the unlabeled data X_u to retrain a new SVM. The



Semi-Supervised Learning. Figure 1. Shows the effect of using unlabeled data in semi-supervised classification. The class boundary of the generative model-based classifier (d) trained using both the labeled and unlabeled data are quite different from the one trained using just the labeled data (c) [14].



Semi-Supervised Learning. Figure 2. Shows the effect of using unlabeled data in learning a SVM classifier. The *dotted line* shows the classifier trained on only the labeled training data, while the *solid line* is the classifier trained on both labeled and unlabeled data [14].

process is repeated iteratively, increasing the weights of the unlabeled data in each iteration, thereby forcing the classifier decision boundary to pass through the low-density regions of X_u (as shown in Fig. 2).

3. *Graph-based methods.* These approaches encode the data as nodes in a graph (labeled data instances have class labels associated with the nodes), with edges encoding pairwise distance between data instances. They operate under the assumption that the original (high-dimensional) data can be projected to a low-dimensional manifold without significant loss of information, and use the unlabeled points for different types of regularization, e.g., classify unlabeled points using label propagation algorithms that prefer smoothness of inferred classes across edges [15].

There are various other methods of semi-supervised learning that employ different techniques of using unlabeled data to improve supervised learning, e.g., co-training [2], semi-supervised manifold embedding [1], which are outlined in the book and surveys on semi-supervised learning mentioned in the historical background section.

Key Applications

Semi-supervised learning has been applied successfully to various applications in text analysis (e.g., document categorization), natural language processing (e.g., word sense disambiguation), bioinformatics (e.g., protein classification, protein function prediction) image analysis (e.g., handwritten digit/character recognition, face

detection, facial expression recognition), and speech recognition (e.g., speaker identification).

Cross-references

- ▶ Classification
- ▶ Clustering with Constraints

Recommended Reading

1. Belkin M. and Niyogi P. Semi-supervised learning on manifolds. Technical Report, The University of Chicago, TR-2002-12, 2002.
2. Blum A. and Mitchell T. Combining labeled and unlabeled data with co-training. In Proc. 11th Annual Conf. on Computational Learning Theory, 1998, pp. 92–100.
3. Chapelle O., Schölkopf B., and Zien A. (eds.). Semi-supervised learning. MIT Press, Cambridge, MA, 2006.
4. Collins M. and Singer Y. Unsupervised models for named entity classification. In Proc. Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora, 1999.
5. Dempster A.P., Laird N.M., and Rubin D.B. Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc. B, 39:1–38, 1977.
6. Hosmer D.W. Jr. A comparison of iterative maximum likelihood estimates of the parameters of a mixture of two normal distributions under three different types of sample. Biometrics, 29(4):761–770, 1973.
7. Joachims T. Transductive inference for text classification using support vector machines. In Proc. 16th Int. Conf. on Machine Learning, 1999, pp. 200–209.
8. Nigam K., McCallum A., Thrun S., and Mitchell T. Learning to classify text from labeled and unlabeled documents. In Proc. 11th National Conf. on AI, 1998, pp. 792–799.
9. Ratsaby J. and Venkatesh S.S. Learning from a mixture of labeled and unlabeled examples with parametric side information. In Proc. Eighth Annual Conf. on Computational Learning Theory, 1995, pp. 412–417.
10. Scudder H.J. Probability of error of some adaptive pattern-recognition machines. IEEE Trans. Inf. Theory, 11:363–371, 1965.
11. Seeger M. Learning with labeled and unlabeled data. Technical Report, Edinburgh University, 2001.
12. Vapnik V.N. and Chervonenkis A. Theory of pattern recognition [in Russian]. Nauka, Moscow, 1974.
13. Yarowsky D. Unsupervised word sense disambiguation rivaling supervised methods. In Proc. 23rd Annual Meeting of the Assoc. for Computational Linguistics, 1995, pp. 189–196.
14. Zhu X. Semi-supervised learning literature survey. Computer Sciences Technical Report TR 1530, University of Wisconsin Madison, 2006.
15. Zhu X., Ghahramani Z., and Lafferty J. Semi-supervised learning using Gaussian fields and harmonic functions. In Proc. 20th Int. Conf. on Machine Learning, 2003.

Sense and Respond Systems

- ▶ Event Driven Architecture

Sensitivity

- ▶ Precision and Recall

Sensor Network Systems

- ▶ Event Driven Architecture

Sensor Networks

RAMESH GOVINDAN

University of Southern California, Los Angeles,
CA, USA

Synonyms

Wireless sensor networks; Embedded networked sensing; Sensorsnet

Definition

A sensor network is a collection of small battery-operated embedded devices each with a built-in processor, sensors, and wireless communication capability. It usually has one or more *gateways*, or embedded computers, connected to an external communications infrastructure (e.g. the Internet or the cellular network). Either the embedded devices or the gateways may be mobile. The purpose of a sensor network is to collaboratively sense the environment, and convey (a possibly condensed version of) the sensed information to one or more human users via the gateways. More advanced sensor networks may autonomously react to the sensed information and effect an action, such as activating a camera or a light switch.

Historical Background

As early as 1977, researchers appear to have discussed, in significant depth, the various issues in building a distributed wireless sensor network. These discussions were part of the ARPA Sensor Net project, and foreshadowed many of the issues that sensor network researchers have been more recently addressing. Moreover, with great prescience, project participants enumerated many application scenarios that are relevant even today: airborne deployment of sensors, wireless surveillance, tracking, and so on.

Perhaps the first concrete instance of a wireless sensor node was the WINS prototype developed by Pottie and Kaiser [2]. The development of these prototypes made significant advances in low-power electronics and device integration. Moreover, the WINS project anticipated the need for sophisticated network self-configuration in wireless sensor networks.

However, sensor networking really blossomed as a separate discipline following the “Simple Systems” DARPA ISAT study led by Deborah Estrin, and the subsequent establishment of a DARPA-funded program networked sensing called SenseIT. The SenseIT program explored a number of important early research directions in the field, including collaborative signal processing, energy-efficient data dissemination, and the application of database concepts to data management in sensor networks.

Foundations

The technological constraints imposed by battery-operated wireless sensors have resulted in five categories of research questions that have been explored by the sensor networks community:

1. *Energy-awareness.* Since sensor nodes operate on batteries, the lifetime of a sensor network is determined by how quickly individual sensor nodes drain their energy resources. This line of research examines techniques for increasing network lifetime through careful energy management or harvesting.
2. *Communication reliability.* Wireless communication is notoriously unpredictable, and sensor networks are often deployed in heavily obstructed environments. As such, being able to reliably retrieve sensed data, while still being energy-efficient, is a significant challenge.
3. *Spatio-temporal awareness.* Information generated by sensors is useful only when it is associated with a spatio-temporal context that indicates when and where the data was generated. To do this, sensor nodes need accurate positioning and time synchronization technologies. Unfortunately, these nodes are often deployed in locations (indoors, or in foliage) where the Global Positioning System (GPS) is unavailable.
4. *Programmability.* Sensor networks have a very large number of potential applications, and there is a dire need for novel programming paradigms that

hide the complexity imposed by the previous challenges, yet promote reusable and easily understood systems.

5. *Security and Data integrity.* Because sensors may be deployed in unattended and harsh environments, there is critical need for software that ensures the security of the data generated by the sensors, and the overall integrity or correctness of the data.

The following paragraphs summarize research in these areas.

Energy-awareness. Perhaps most widely studied, research in improving network lifetime has proceeded on two fronts. There has been exploratory work on technologies for harvesting energy from light sources, from vibration, or by using mobility to find a nearby power source. This line of research aims to renew the supply of energy at a node. More extensively studied is the direction in which careful algorithms and systems design techniques are used to conserve energy. Specifically, it has been experimentally determined that communication and sensing are among the primary contributors to energy usage. To conserve communication energy, researchers have explored techniques like *data aggregation* [6,11] where sensor readings are processed and possibly compressed within the network before being transmitted. In addition, techniques employed at various software layers (medium access, routing, and application) that turn the radio off or put the node to sleep during periods of inactivity have also been studied. To conserve sensing energy, researchers have proposed exploiting correlations between sensors, or using one sensor to trigger another higher-energy sensor.

Communication reliability. Wireless packet transmissions are susceptible to losses or corruption due to environmental noise and collisions from concurrent transmissions. To overcome losses due to packet corruption, researchers have explored careful coding techniques that allow for packet reconstruction using relatively small amounts of information. Another line of research has explored the construction of high-quality routing paths which attempt to reduce the likelihood of corruption [15]. To ensure reliable end-to-end delivery, researchers have explored the combined use of hop-by-hop recovery and end-to-end retransmissions. To reduce packet loss due to contention, a line of work has explored *congestion control* techniques that dynamically adapt a node's sending rate to avoid saturating the channel

capacity [13]. These techniques essentially sample the current channel conditions, and adjust node sending rates according to feedback control laws that ensure stability and efficiency.

Spatio-temporal awareness. When a sensor node is deployed, it faces two important questions: "Where am I?" and "What is the time?" Without answers to these questions, the data generated by sensor networks becomes meaningless. Since GPS cannot be assumed to work well in the kinds of obstructed environments that sensor networks are likely to be deployed in, researchers have explored a wide variety of methods for *localization* and *time synchronization*. For the former problem, a variety of devices (such as ultrasound, radio, lasers) have been used to estimate the distances between two nodes, and a variety of techniques ranging from multi-lateration and least-squares regression to multidimensional scaling have been used to take these estimates and place nodes in a coordinate system [9]. For the latter problem, a similar approach has been followed: methods to locally synchronize clocks between neighboring nodes use message transmission and processing latencies and carefully compensate for clock drift and skew [12]; a second class of methods attempts to synchronize clocks across the entire network while minimizing error accumulation at every hop.

Programmability. Programming these tiny sensors because of the platform constraints imposed by form factor and lifetime requirements: energy, processor, memory, and wireless communication bandwidths are all constraints that affect application development. The literature on programming sensor networks has focused on how to relieve the burden of the programmer in dealing with these constraints. Starting from seminal work on event-driven operating systems [5], researchers have moved on to higher-level abstractions that simplify programming. These include specialized abstractions for programming individual nodes such as virtual machines [10], state machines, neighborhood communication abstractions, and so on. More recently, researchers have turned towards specialized programming languages for expressing the behavior of the network as a whole [8]. An important thread in this line of research is the application of database techniques to program sensor networks: this thread is discussed below.

Security and data integrity. More recently, research in sensor networks has turned towards techniques to ensure the integrity of data produced by the sensors

and transmitted across the network. Fundamentally, sensors can produce erroneous results, and the data transmitted by sensors can be inadvertently or maliciously corrupted while in transit. Techniques to address these problems have examined encryption and authentication mechanisms for medium access [7] and routing, as well as statistical techniques for detecting outliers in sensor data or for being able to quantify the confidence attributable to a received result.

Data-Centric Techniques in Sensor Networks

An important development in the history of sensor networking has been the emergence of *data-centricity* – the use of programming abstractions that specify attributes or characteristics of data, rather than the nodes at which data may be found. Two views of data-centricity in sensor networks have emerged: the *database view* which views the network as a virtual relational table, and the networking view which views the network as a distributed hash table [14].

More prominently explored, research on the database view has been motivated by three challenges: first, the data being generated by sensors is continuously changing; second, because of quantization effects and variable spatial density, these data are inherently approximate; and third, given the energy constraints, it is infeasible to extract large volumes of data from the network. These constraints have motivated research on approximate, aggregate, continuous queries, on the virtual relational table resulting from data generated by the various sensors. This setting provides ample scope for exploring query optimization techniques. These optimization techniques exploit user-specified bounds on result error, correlations among different types of sensors, or models of the sensor field in order to reduce communication and sensing cost.

The networking view has explored one-shot point and enumeration queries by modeling the sensor network as a distributed hash table. This approach uses random or locality preserving hashing of generated sensor data to a geographic location in a two-dimensional coordinate system. The data are stored at the corresponding location. This approach enables the construction of innovative distributed indexing structures, such as, for example, those that support multidimensional range queries. Overall, this approach tends to have higher communication costs except in regimes with relatively high query rates (such as, for example, queries generated by programs running within the network).

Key Applications

The following are some examples of potential application areas for sensor networks.

1. *Atmospheric.* In this application, sensors measure the concentration of pollutants or carbon dioxide, as well as other atmospheric conditions (temperature, humidity, wind direction and speed). These dense measurements can give a much clearer picture of local variations in atmospheric conditions.
2. *Habitats.* Sensors can be used to measure light, temperature, humidity, and photo-synthetically active radiation across a forest floor or a forest canopy transect. This gives a detailed picture of spatio-temporal variations in these quantities, and can help biologists understand the effect of these variations on local distributions of plant and animal life.
3. *Water.* Sensor nodes deployed on buoys on the surface of the lake or near the seashore and measuring temperature gradients and chlorophyll beneath the lake's surface can help marine biologists study the dynamics of plankton populations. These populations can significantly affect marine life and thereby have huge economic impact.
4. *Soil.* A network of sensors deployed in soil can measure temperature, light, humidity and contaminant flow. Such measurements can aid precision agriculture for improving crop yields and ensuring better land management.
5. *Man-made structures.* Integrity and energy-expenditure of structures. Sensors deployed on buildings and measuring light and temperature conditions, as well as ambient vibrations, can help understand (and control) energy usage in large buildings as well as assess the integrity of these structures. Similar sensor networks can also be used to measure activity in seismically active areas (e.g., volcanoes).

Future Directions

In the first 5–8 years of its existence, sensor networks were driven by a technological push. Advances in miniaturization made it possible to envision networks of small devices, and these advances prompted many of the research directions described above. Now that the potential of sensor networks is well established, the next phase of research has to deliver reliable, manageable systems that provide ease of programmability. Thereafter, advances in sensor networks will be driven by experiences obtained from large-scale deployments.

Cross-references

- ▶ Continuous Queries in Sensor Networks
- ▶ Data Acquisition and Dissemination in Sensor Networks
- ▶ Data Aggregation in Sensor Networks
- ▶ Data Compression in Sensor Networks
- ▶ Data Estimation in Sensor Networks
- ▶ Data Fusion in Sensor Networks
- ▶ Data Storage and Indexing in Sensor Networks
- ▶ Database Languages for Sensor Networks
- ▶ Query Optimization in Sensor Networks

Recommended Reading

1. Akyildiz I., Su W., Sankarasubramaniam Y., and Cayirci E. A survey on sensor networks. *IEEE Commun. Mag.*, 40(8):102–114, 2002.
2. Asada G., Dong T., Lin F., Pottie G., Kaiser W., and Marcy H. Wireless integrated network sensors: low power systems on a chip. In Proc. European Solid State Circuits Conference, 1998.
3. Culler D.E. and Hong W. Wireless sensor networks – introduction. *Commun. ACM*, 47(6):30–33, 2004.
4. Estrin D., Govindan R., and Heidemann J. Embedding the internet: introduction. *Commun. ACM*, 43(5):38–41, 2000.
5. Hill J., Szewczyk R., Woo A., Hollar S., Culler D., and Pister K. System architecture directions for networked sensors. *SIGPLAN Not.*, 35(11):93–104, 2000.
6. Intanagonwiwat C., Govindan R., and Estrin D. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In Proc. 6th Annual Int. Conf. on Mobile Computing and Networking, 2000, pp. 56–67.
7. Karlof C., Sastry N., and Wagner D. TinySec: link-layer encryption for tiny devices. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 162–175.
8. Kothari N., Gummadi R., Millstein T., and Govindan R. Reliable and efficient programming abstractions for wireless sensor networks. In Proc. SIGPLAN Conf. on Programming Language Design and Implementation, 2007, pp. 10–13.
9. Langendoen K. and Reijers N. Distributed Localization in Wireless Sensor Networks: A Quantitative Comparison. Technical Report PDS-2002-003, Technical University, Delft, November 2002.
10. Levis P. and Culler D. Maté: a tiny virtual machine for sensor networks. In Proc. 10th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 2002, pp. 85–95.
11. Madden S., Franklin M.J., Hellerstein J.M., and Hong W. TAG: Tiny AGgregate queries in ad-hoc sensor networks. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002, pp. 131–146.
12. Maróti M., Kusy B., Simon G., and Lédeczi Á. The flooding time synchronization protocol. In Proc. 2nd Int. Conf. on Embedded Networked Sensor Systems, 2004, pp. 39–49.
13. Rangwala S., Gummadi R., Govindan R., and Psounis K. Interference-aware fair rate control in wireless sensor networks.

In Proc. ACM SIGCOMM Symp. on Network Architectures and Protocols, 2006.

14. Woo A., Madden S., and Govindan R. Networking support for query processing in sensor networks. *Commun. ACM*, 47(6):47–52, 2004.
15. Woo A., Tong T., and Culler D. Taming the underlying challenges of reliable multihop routing in sensor networks. In Proc. 1st Int. Conf. on Embedded Networked Sensor Systems, 2003.

Sensornet

- ▶ Sensor Networks

Sentiment Analysis

- ▶ Opinion Mining

SEQUEL

- ▶ SQL

Sequence Data Mining

- ▶ Temporal Data Mining

Sequenced Semantics

MICHAEL H. BÖHLEN¹, CHRISTIAN S. JENSEN²

¹Free University of Bozen-Bolzano, Bolzano, Italy

²Aalborg University, Aalborg, Denmark

S

Definition

Sequenced semantics make it possible to generalize a query language statement on a non-temporal database to a temporal query on a corresponding temporal, interval-timestamped database by applying minor syntactic modifications to the statement that are independent of the particular statement. The semantics of such a generalized statement is consistent with considering the temporal database as being composed of a sequence of non-temporal database states. Sequenced

semantics takes into account the interval timestamps of the argument tuples when forming the interval timestamps associated with result tuples, as well as permits the use of additional timestamp-related predicates in statements.

Key Points

A question that has intrigued temporal database researchers for years is how to systematically generalize non-temporal query language statements, i.e., queries on non-temporal databases, to apply to corresponding temporal databases. A prominent approach is to view a temporal database as a sequence of non-temporal databases. Then a non-temporal statement is rendered temporal by applying it to each non-temporal database, followed by integration of the non-temporal results into a temporal result. Sequenced semantics formalizes this approach and is based on three concepts: S-reducibility, extended S-reducibility, and interval preservation. These topics are discussed in turn.

The ensuing examples assume a database instance with three relations:

Employee

ID	Name	VTIME
1	Bob	5–8
3	Pam	1–3
3	Pam	4–12
4	Sarah	1–5

Salary

ID	Amt	VTIME
1	20	4–10
3	20	6–9
4	20	6–9

Bonus

ID	Amt	VTIME
1	20	1–6
1	20	7–12
3	20	1–12

S-Reducibility

S-reducibility states that the query language of the temporally extended data model must offer, for each query q in the non-temporal query language, a *syntactically similar temporal query q^t* that is its natural generalization, i.e., q^t is snapshot reducible to q , and q^t is syntactically identical to $S_1 q S_2$. The goal is to make the semantics of temporal queries easily understandable in terms of the semantics of the corresponding non-temporal queries. The strings S_1 and S_2 are independent of q and are termed *statement modifiers* because they change the semantics of the entire statement q that they enclose.

In the following examples, statements are prefixed with the modifier SEQ VT [2]. This modifier tells the temporal DBMS to evaluate statements with sequenced semantics in the valid-time dimension. These examples illustrate that S-reducible statements are easy to write and understand because they are simply conventional SQL statements with the additional prefix SQL VT. Writing statements that compute the same results, but without using statement modifiers, can be very difficult [3].

```
SEQ VT SELECT * FROM EMPLOYEE;
SEQ VT
  SELECT ID
  FROM EMPLOYEE AS E
  WHERE NOT EXISTS (
    SELECT *
    FROM SALARY AS S
    WHERE E.ID = S.ID);
```

The first query returns all `Employee` tuples together with their valid time – this corresponds to returning the content of `Employee` at each state. The second query determines the time periods when an employee did not get a salary. It returns $\{(3, 1–3), (3, 4–5), (3, 10–12), (4, 1–5)\}$. Conceptually the enclosed statement is evaluated on each state of the database. Computationally, the interval 6–9 is subtracted from the interval 4–12 to get the intervals 4–5 and 10–12.

Extended S-Reducibility

S-reducibility is applicable only to queries of the underlying non-temporal query language and does not extend to queries with explicit references to time. Consider the following queries.

```

SEQ VT
SELECT E.ID
FROM Employee AS E, Salary AS S
WHERE E.ID = S.ID
AND DURATION(VTIME(E)) >
DURATION(VTIME(S));

```



```

SEQ VT
SELECT E.ID, VTIME(S), VTIME(E)
FROM Employee AS E, Salary AS S
WHERE E.ID = S.ID;

```

The first query constrains the temporal join to tuples in `Employee` with a valid time that is longer than the valid time of the salary tuple it shall be joined with. This condition cannot be evaluated on individual non-temporal relation states because the timestamp is not present in these states. Nevertheless, the temporal join itself can still be conceptualized as a non-temporal join evaluated on each snapshot, with an additional predicate. The second query computes a temporal join as well, but also returns the original valid times. Again, the semantics of this query fall outside of snapshot reducibility because the original valid times are not present in the non-temporal relation states.

DBMSs generally provide predicates and functions on time attributes, which may be applied to, e.g., valid time, and queries such as these arise naturally. Applying sequenced semantics to statements that include predicates and functions on time offers a higher degree of orthogonality and wider ranging temporal support.

Interval Preservation

Coupling snapshot reducibility with syntactical similarity and using this property as a guideline for how to semantically and syntactically embed temporal functionality in a language is attractive. However, S-reducibility does not distinguish between different relations if they are snapshot equivalent. This means that different results of an S-reducible query are possible: the results will be snapshot equivalent, but will differ in how the result tuples are timestamped. As an example, consider a query that fetches and displays the content of the `Bonus` relation. An S-reducible query may return the result $\{\langle 1, 20, 1-6 \rangle, \langle 1, 20, 7-12 \rangle, \langle 3, 20, 1-12 \rangle\}$. If Bob received a 20K bonus for his performance during the first half of the year and another 20K bonus for his performance during the second half of the year and Pam received a 20K bonus for her

performance during the entire year, this is the expected result. This is also the result supported by the three tuples in the example instance displayed above. However, S-reducibility does not distinguish this result from any other snapshot equivalent result. With S-reducibility a perfectly equivalent result would be $\{\langle 1, 20, 1-12 \rangle, \langle 3, 20, 1-6 \rangle, \langle 3, 20, 7-12 \rangle\}$.

Interval preservation settles the issue of which result should be favored out of the many possible results permitted by S-reducibility. When defining how to timestamp tuples of query results, two possibilities come to mind. Results can be coalesced. This solution is attractive because it defines a canonical representation for temporal relations. A second possibility is to consider lineage and preserve, or respect, the timestamps as originally entered into the database [1]. Sequenced semantics requires that the default is to preserve the timestamps – being irreversible, coalescing cannot be the default.

Cross-references

- ▶ [Nonsequenced Semantics](#)
- ▶ [Snapshot Equivalence](#)
- ▶ [Temporal Coalescing](#)
- ▶ [Time Interval](#)
- ▶ [Valid Time](#)

Recommended Reading

1. Böhlen M.H., Busatto R., and Jensen C.S. Point- versus interval-based temporal data models. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 192–200.
2. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. ACM Trans. Database Syst., 25(4):48, 2000.
3. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann, 1999.

Sequential Patterns

JIANYONG WANG

Tsinghua University, Beijing, China

Synonyms

[Frequent subsequences](#)

Definition

A sequence database $D = \{S_1, S_2, \dots, S_n\}$ for sequential pattern mining consists of n input sequences (where $n \geq 1$), and an input sequence

$S_i = \langle e_{i_1}, e_{i_2}, \dots, e_{i_m} \rangle$ ($1 \leq i \leq n$) is an ordered list of m events (where $m \geq 1$). Each event e_{ij} ($1 \leq i \leq n, 1 \leq j \leq m$) is a non-empty set of items. Given two sequences, $S_a = \langle e_{a_1}, e_{a_2}, \dots, e_{a_k} \rangle$ and $S_b = \langle e_{b_1}, e_{b_2}, \dots, e_{b_l} \rangle$, if $k \leq l$ and there exist integers $1 \leq x_1 < x_2 < \dots < x_k \leq l$ such that $e_{a_1} \subseteq e_{b_{x_1}}, e_{a_2} \subseteq e_{b_{x_2}}, \dots, e_{a_k} \subseteq e_{b_{x_k}}$, S_b is said to contain S_a (or equivalently, S_a is said to be contained in S_b). The number of input sequences in D that contain sequence S is called the *support* of S in D , denoted by $\text{sup}^D(S)$. Given a user-specified minimum support threshold min_sup , S is called a *sequential pattern* (or a *frequent subsequence*) in D if $\text{sup}^D(S) \geq \text{min_sup}$. If there exists no proper supersequence of a sequential pattern S with the same support as S , S is called a *closed sequential pattern* (or a *frequent closed subsequence*) in D . Furthermore, a sequential pattern S is called a *maximal sequential pattern* (or a *frequent maximal subsequence*) if it is not contained in any other sequential pattern. The problems of *sequential pattern mining*, *closed sequential pattern mining*, and *maximal sequential pattern mining*, are to find all frequent subsequences, all frequent closed subsequences, and all frequent maximal subsequences from input sequence database D , respectively, given a user-specified minimum support threshold min_sup .

Historical Background

Similar to association rule mining, sequential pattern mining was initially motivated by the decision support problem in retail industry and was first proposed by Rakesh Agrawal and Ramakrishnan Srikant in [1]. Later on, it was applied to other domains. Some recent research work further validated its utility in various applications, such as identifying outer membrane proteins, automatically detecting erroneous sentences, discovering block correlations in storage systems, identifying copy-paste and related bugs in large-scale software code, API specification mining and API usage mining from open source repositories, frequent subsequence-based XML document clustering, sequence-based XML query pattern mining for effective caching, and Web log data mining.

In the seminal paper on sequential pattern mining, three algorithms were introduced [1]. Among these algorithms, AprioriSome and DynamicSome were proposed for mining maximal sequential patterns, while AprioriAll was designed for mining all sequential patterns. The same authors later generalized the sequential pattern mining problem by allowing time constraints,

sliding time window, and taxonomies, and proposed a new algorithm, GSP [10]. The inefficiency of AprioriAll mainly stems from its computationally expensive data transformation operation which transforms each transaction to a set of frequent itemsets in order to find sequential patterns. As GSP overcomes the drawbacks of AprioriAll, it is much faster than AprioriAll. In [4], Jiawei Han et al. proposed the FreeSpan algorithm, which shows better performance than GSP. In [15], Mohammed J. Zaki adopted the vertical data representation for sequential pattern mining and devised an efficient algorithm, SPADE, which fully exploits the lattice search techniques and some join operations. Another state-of-the-art sequential pattern mining algorithm is PrefixSpan, which was proposed by Jian Pei et al. [8]. It adopts a projection-based, sequential pattern growth approach to avoiding the traditional candidate-generation-and-test paradigm, thus improves the algorithm efficiency. In [3], Jay Ayres et al. designed another sequential pattern mining algorithm, SPAM. This algorithm integrates a depth-first search strategy with some effective pruning techniques, uses a vertical bitmap data representation, and can incrementally output frequent subsequences in an online fashion.

All the preceding algorithms except AprioriSome and DynamicSome mine the complete set of sequential patterns. One problem with sequential pattern mining is that it may generate too many redundant patterns, which also impedes the algorithm efficiency. One popular solution to this problem is to mine closed sequential patterns only, which usually leads to not only a more compact yet complete result set but also better efficiency. In [14], Xifeng Yan and Jiawei Han presented the CloSpan approach, which incorporates several effective pruning methods into the PrefixSpan framework and achieves much better performance than PrefixSpan. Recently, another closed sequential pattern mining algorithm, BIDE, was proposed in [12] by Jianyong Wang et al. It integrates a new pattern closure checking scheme and a new pruning technique with the PrefixSpan framework, and is both runtime and memory efficient.

Foundations

The biggest challenge faced by sequential pattern mining is the combinatorial explosion problem. To alleviate this problem, researchers have tried various ways. In the following some factors which may have an impact on the efficiency of a sequential pattern mining (or closed sequential pattern mining) algorithm are

summarized. These factors mainly include the data representation format, pattern enumeration framework (i.e., search strategy), search space pruning techniques, and pattern closure checking scheme.

Sequence Data Format

The input sequence data can be represented in two alternative formats. The *horizontal representation* is a natural bookkeeping of the input sequences. Each sequence consists of an ordered list of events, while each event is recorded as a list of items (which are supposed in most algorithms to be sorted according to a certain order, say the lexicographical order). AprioriAll, GSP, PrefixSpan, CloSpan, and BIDE are typical examples adopting the horizontal representation. Note that AprioriAll needs to first find the frequent itemsets and transform each event to the set of frequent itemsets contained in the event in order to find sequential patterns. To compute the support of a subsequence, the horizontal format based algorithms need to scan the database, which is computationally expensive. To assist support counting, these algorithms devise some special mechanisms. For example, GSP introduces the hash-tree data structure, while PrefixSpan, CloSpan, and BIDE use a projection-based approach to shrink the part of database that needs to be scanned.

In the *vertical representation*, the database is represented as a set of items, where each item is recorded as a set of pairs of sequence identifier (SID) and event identifier (EID) containing the item. SPADE and SPAM use the vertical format. With the vertical representation, the support-counting can be performed using simple join operations with temporal ordering constraint. To improve the efficiency of support counting, SPAM proposes to use vertical bitmaps to represent the sequence database. Each item is converted to a bitmap, which has a bit for each event in the database. If an event contains an item, the corresponding bit regarding the event and item is set to one; otherwise, it is set to zero. Based on the transformed bitmap sequence representation, efficient support counting can be easily achieved using bitwise AND operations of bitmaps.

Search Strategy

Given an input sequence database D and a minimum support threshold min_sup , the set of sequential patterns are deterministic, and can be organized into a lexicographic frequent sequence tree structure. Suppose $min_sup=2$, database D contains three input

sequences, $\langle (A \ B \ D)(B \ C \ D)(A) \rangle$, $\langle (B) \ (A \ B \ E) \ (B) \rangle$, and $\langle (A \ B) \ (B \ C \ D) \rangle$ (here the commas separating each pair of adjacent events in the same sequence are omitted), respectively, and there exists a lexicographic ordering among the set of distinct items $A \leq B \leq C \leq D \leq E$. For a prefix subsequence S_p , it can be extended in two ways, namely, sequence-extension and itemset-extension. The *sequence-extension* extends S_p by a new event containing a single item, while *itemset-extension* adds a new item to the last event of S_p and the new item must be lexicographically larger than any item of the last event of S_p . Assume both sequence-extension and itemset-extension are performed in lexicographic ordering and itemset-extension is performed before sequence-extension for the same prefix sequence. Then, the lexicographic sequence tree structure of the frequent subsequences in the running example is shown in Fig. 1. Note that each node in the tree shows a sequential pattern and its corresponding support (i.e., the number after the colon), and all the patterns at the same level have the same length (namely, they contain the same number of items). An edge which links a parent node P at level k to a child node C at level (k+1) indicates that the pattern at C is directly extended from the prefix pattern at node P, either by sequence-extension or by itemset-extension.

Once the sequential patterns are organized into a lexicographic tree structure, one can choose a tree traversal strategy for sequential pattern mining. The two popular search strategies are breadth-first search and depth-first search. In the *breadth-first search* method, frequent subsequences are mined in a level-wise manner, that is, before mining patterns with length (k+1), one needs to first mine all patterns with length k. In contrast, a *depth-first search* method traverses the sequence tree in depth-first order. GSP adopts the breadth-first search strategy to enumerate the sequential patterns, PrefixSpan, SPAM, CloSpan, and BIDE choose the depth-first search paradigm, while SPADE supports both breadth-first search and depth-first search methods.

Search Space Pruning

One of the most crucial optimization considerations to improve the efficiency of a data mining algorithm is to devise some effective search space pruning techniques. Based on some heuristics, if some parts of the search space are already known to be futile in generating sequential patterns, they should be found and pruned as quickly as possible. Perhaps the most

well-known property used for designing pruning methods in frequent pattern mining is the *Apriori* property (also known as the downward closure property or anti-monotone property). It states in the sequence mining setting that all the subsequences of a frequent sequence must be also frequent, or equivalently, a sequence must be infrequent if it contains an infrequent subsequence. All the existing sequential pattern mining algorithms have exploited the Apriori property in different ways.

From Fig. 1 one can see that only three out of the 31 frequent subsequences are closed, namely, (B)(A):2, (AB)(B):3, and (AB)(BCD):2, and many subtrees in Fig. 1 contain no closed sequential pattern. An efficient closed sequential pattern mining algorithm should avoid traversing the subtrees containing no closed patterns, which leaves room to further prune the search space. Both CloSpan and BIDE adopt some optimization techniques. The pruning methods proposed in CloSpan are listed as follows.

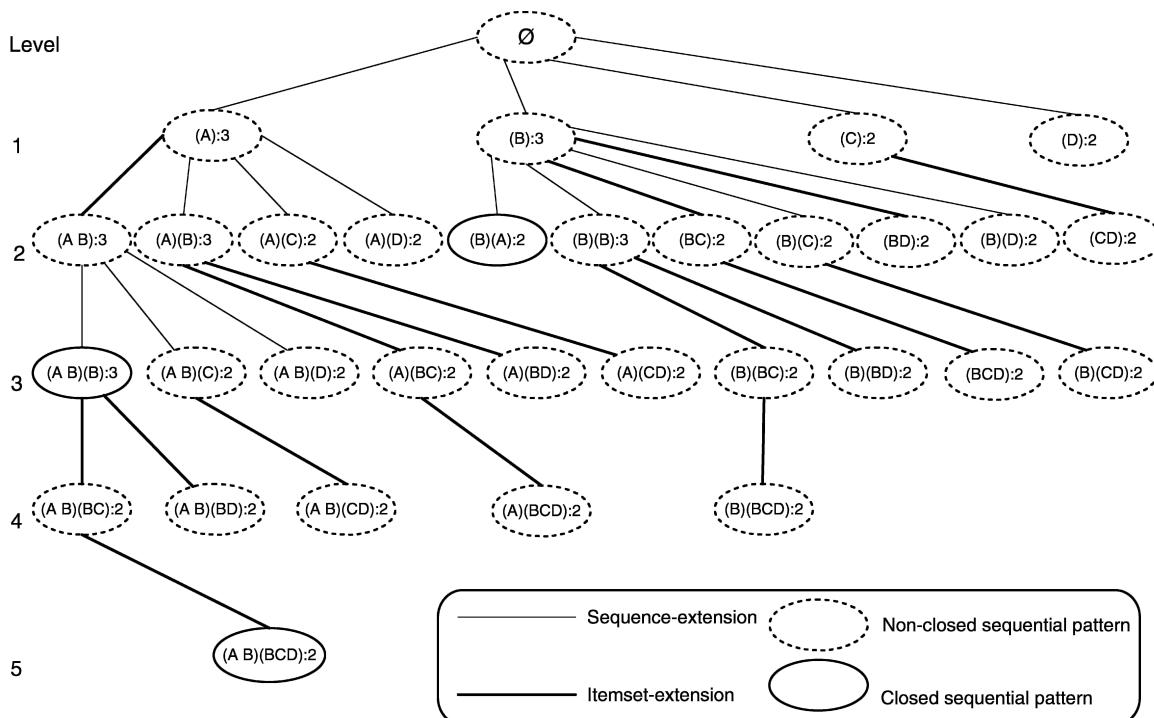
- *Common prefix pruning*: if there exists a common prefix, all sequences beginning with a proper subsequence of this prefix cannot be closed.

- *Partial-order pruning*: if an item “a” always occurs before item “b” in all sequences, any sequence beginning with “b” cannot be closed.
- *Equivalent projected DB pruning*: given two subsequences, s and s' , where s is a proper subsequence of s' and they have equivalent projected database, any sequence beginning with s cannot be closed.

The concepts of common prefix, partial-order, and equivalent projected database can be found in [7]. The BIDE algorithm proposes a single but effective pruning technique called *BackScan* search space pruning. Given a prefix S_p if $\exists i$ (i is a positive integer and is no greater than the length of S_p) and there exists any item that appears in each of its i th semimaximum periods, S_p can be safely pruned. The interested readers are referred to [12] for more details.

Pattern Closure Checking Scheme

The optimization methods proposed in CloSpan and BIDE are very effective in pruning the unpromising parts of the search space, however, they cannot assure that each discovered sequential pattern is closed. For closed sequential pattern mining algorithms, one still



Sequential Patterns. Figure 1. The lexicographic sequence tree structure of the frequent subsequences in the running example.

needs to devise some methods to check if a sequential pattern is closed or not. In CloSpan, all the candidate closed sequential patterns are maintained in a tree data structure. CloSpan eliminates the non-closed patterns in a post-processing phase and adopts the hashing technique to accelerate the pattern closure checking. When the number of candidate sequential patterns is large, the pattern tree structure may consume non-trivial space. In [12], BIDE adopts a so-called *BI-Directional Extension* closure checking scheme. One big advantage of this new scheme is that it avoids maintaining the set of candidate closed sequential patterns, and thus saves space. If there exists no *forward-S-extension* item, *forward-I-extension* item, *backward-S-extension* item, nor *backward-I-extension* item with respect to a prefix sequence S_p , S_p is a closed sequence, otherwise, S_p must be non-closed. The definitions of a forward-S-extension item, a forward-I-extension item, a backward-S-extension item, and a backward-I-extension item can be found in [8].

Key Applications

In recent years sequential pattern mining witnessed many applications, which roughly fall into the following categories.

Frequent Subsequence-based Classifier

In [9], the authors used an efficient implementation of generalized suffix tree to mine a set of frequent subsequences with a minimum length constraint, and built the rule-based classifier and SVM classifier based on the discovered frequent subsequences. Their performance results demonstrate that the frequent subsequence-based classifier achieves high accuracy in identifying outer membrane proteins. Recently the authors of [11] proposed a method to build associative classification rules from frequent subsequences returned by a variant of the PrefixSpan algorithm. Their performance study shows that sequence-based classification rules are very helpful in automatically detecting erroneous sentences.

Operating System and Software Engineering

In [5,6], the authors adopted the CloSpan algorithm to mine closed sequential patterns, which have been shown very useful in discovering block correlations in storage systems and identifying copy-paste and related bugs in large-scale software code. In [7,13], the BIDE algorithm was used to discover the set of frequent closed subsequences with the purpose of API

specification mining and API usage mining from open source repositories.

Frequent Subsequence-based XML Data Management

Sequential pattern mining has also been widely applied in semi-structured data management. In this application, an XML document is first converted to a sequence instead of a tree structure. Then some kinds of constrained frequent subsequences are mined, which can be exploited to accelerate XML query or XML document clustering. The performance results in [2] demonstrate that the frequent subsequence-based XML clustering algorithm XProj achieves better clustering quality than previous algorithms.

Web Log Data Mining

Some researchers have also applied sequential pattern mining algorithms in mining salient patterns (e.g., contiguous sequential patterns) from Web log data.

Experimental Results

Some experimental results on sequential pattern mining can be found in [3,8,10,15], which compares the efficiency of some typical sequential pattern mining algorithms including GSP, SPADE, PrefixSpan, and SPAM, while [14] and [12] present the performance study of two closed sequential pattern mining algorithms, CloSpan and BIDE.

Data Sets

The IBM synthetic dataset generator can generate sequence datasets and can be found from the link of <http://www.cs.rpi.edu/~zaki/software/>. The popularly used real sequence datasets include some Web log data, and protein sequence datasets.

URL to Code

The code for PrefixSpan and CloSpan algorithms can be found from the Illini Mine portal, <http://dm1.cs.uiuc.edu/protected/im>, the code for SPADE algorithm can be downloaded from the link of <http://www.cs.rpi.edu/~zaki/software/>, while the code for BIDE algorithm can be traced from <http://dbgroup.cs.tsinghua.edu.cn/wangjy/>.

Cross-references

- [Apriori Property and Breadth-First Search Algorithms](#)
- [Closed Itemset Mining and Non-redundant Association Rule Mining](#)

- ▶ Frequent Graph Patterns
- ▶ Frequent Itemsets and Association Rules
- ▶ Frequent Partial Orders
- ▶ Pattern-Growth Methods

Recommended Reading

1. Agrawal R. and Srikant R. Mining sequential patterns. In Proc. 11th Int. Conf. on Data Engineering, 1995.
2. Aggarwal C.C., Ta N., Wang J., Feng J., and Zaki M.J. XProj: a framework for projected structural clustering of XML documents. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007.
3. Ayres J., Gehrke J., Yiu T., and Flannick J. Sequential pattern mining using a bitmap representation. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002.
4. Han J., Pei J., Mortazavi-Asl B., Chen Q., Dayal U., and Hsu M.C. FreeSpan: frequent pattern-projected sequential pattern mining. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000.
5. Li Z., Chen Z., Srinivasan S., and Zhou Y. C-Miner: mining block correlations in storage systems. In Proc. 3rd USENIX Conf. on File and Storage Technologies, 2004.
6. Li Z., Lu S., Myagmar S., and Zhou Y. CP-Miner: finding copy-paste and related bugs in large-scale software code. IEEE Trans. Software Eng., 32(3):176–192, 2006.
7. Lo D. and Khoo S.C. SMArTIC: towards building an accurate, robust and scalable specification miner. In SIGSOFT FSE., 2006.
8. Pei J., Han J., Mortazavi-Asl B., Pinto H., Chen Q., Dayal U., and Hsu M.C. PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern-growth. In Proc. 17th Int. Conf. on Data Engineering, 2001.
9. She R., Chen F., Wang K., Ester M., Gardy J.L., and Brinkman F.S.L. Frequent-subsequence-based prediction of outer membrane proteins. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003.
10. Srikant R. and Agrawal R. Mining sequential patterns: generalizations and performance improvements. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996.
11. Sun G., Liu X., Cong G., Zhou M., Xiong Z., Lee J., and Lin C.Y. Detecting erroneous sentences using automatically mined sequential patterns. In Proc. 45th Annual Meeting of the Assoc. for Computational Linguistics, 2007.
12. Wang J., Han J., and Li C. Frequent closed sequence mining without candidate maintenance. IEEE Trans. Knowl. Data Eng., 19(8):1042–1056, 2007.
13. Xie T. and Pei J. Data mining for software engineering. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006.
14. Yan X., Han J., and Afshar R. CloSpan: mining closed sequential patterns in large databases. In Proc. SIAM International Conference on Data Mining, 2003.
15. Zaki M.J. SPADE: an efficient algorithm for mining frequent sequences. Machine Learn., 42:31–60, 2001.

Serializability

K. VIDYASANKAR

Memorial University of Newfoundland, St. John's,
NL, Canada

Synonyms

Correctness criterion for concurrent executions

Definition

A database is a collection of data items operated on concurrently by several programs. A set of values for the data items is called a *database state*. It is *consistent* if the values satisfy the *integrity constraints* specified for the database. Arbitrary interleaving of the executions of the programs may affect the consistency of the database. The executions that preserve the consistency are called *correct* executions. The notion of serializability helps to identify correct executions. It is based on the transaction concept: a *transaction* is a partially ordered set of atomic steps that constitute an execution of a program, with the property that, when executed alone, it transforms a consistent database state into another consistent one. Examples of atomic steps are read and write of a data item, increment and decrement on a counter object, enqueue and dequeue on a queue object, etc. The sequence of steps in an execution is called a *history*. A *serial* history corresponds to a serial, that is, non-interleaving, execution of the transactions. Any (concurrent) execution whose effect is “equivalent” to that of some serial execution of the same set of transactions is called *serializable*. Clearly, any serial execution of a set of transactions preserves consistency, and hence is correct. Therefore serializable executions are also correct. Serializability theory concerns finding properties of histories which are serializable. This knowledge is useful for designing schedulers that will allow only those concurrent executions that are correct.

Historical Background

After the initial exposure to the problem in [2], several characterizations of serializability were obtained, for example, in [2,6,9,11], and the theory was extended in several directions.

One main direction was defining different notions of serializability corresponding to different interpretations

of ‘equivalence’ to serial histories. Examples are *view-serializability*, where the values read by all the transactions and the final values of all the data items are the same as in some serial execution, *state-serializability* where only the final values of the data items need be the same and the general *S-serializability* where only the values read by a specified subset *S* of the transactions need be the same.

Another main direction was the study of serializability under some constraints [11]. Here, the transaction order in an equivalent serial history must satisfy certain constraints which are usually derived from some syntactic properties of the histories. In general, checking whether a history is serializable is NP-complete. Constraints help to identify subclasses of serializable histories that have a polynomial membership test.

A third direction dealt with relaxing the unit of atomicity. Instead of taking all the steps of a transaction as forming a single unit preserving consistency of the database, groups of several subsets of the steps were considered as atomic units, and interleaving constraints were applied to them. Also, the units of atomicity were allowed to be different for different transactions, giving rise to the *relative atomicity* notion. Hierarchical grouping of the steps led to the study of *nested transactions* [8]. Non-hierarchical groupings were studied, for example, in [3,13].

Another extension was the study of *multi-version serializability* [10]. Here, the value written by each write step of the transactions is kept as a separate version, and a read operation can read any of the versions of the corresponding data item. This notion allows more concurrent executions as correct ones compared to the single version serializability.

Orthogonal developments have been the study and applicability of serializability theory in different architectures (from centralized to distributed databases, multi-databases, mobile environments and recently Internet) and with different objects (from simple data items to complex objects, *B-trees*, design objects, workflows, business processes and Web services). Executions that “slightly” deviate from being serializable and hence are “almost correct” have also been studied. Serializability theory initially focused only on concurrency. Later, recovery aspects were combined and unified theories studied. Details and references can be found in [15].

Foundations

Serializability Characterization

In the following, a simple transaction model is considered where (i) each atomic step is either a *read step* or a *write step*, that reads or writes the value of a data item, respectively, and (ii) each data item is accessed by at most one read step and at most one write step in a transaction, and if both these steps do occur, then the read step occurs before the write step. The theory can easily be extended to other transaction models, and with other atomic steps.

A transaction is *write-only* if it does not have any read steps, and *read-only* if it does not have any write steps. Let *D* be the set of data items in a database, and $\{T_1, \dots, T_n\}$ be a set of transactions. For convenience, a hypothetical write-only initial transaction T_0 which writes the initial values of all the data items, and a hypothetical read-only final transaction T_f which reads the values of all the data items after all the transactions have completed are added. The transaction T_0 helps to set up a consistent initial database state, and T_f helps to argue about the final database state. Let T be $\{T_0, T_1, \dots, T_m, T_f\}$. A read step (write step) of transaction T_i reading (writing) the data item *X* is denoted $R_i[X](W_i[X])$. A set of read steps, unrelated by the partial order, reading a subset *C* of *D* and occurring together, is denoted $R_i[C]$. For example, $R_i[\{X, Y\}]$ denotes the unrelated read steps $R_i[X]$ and $R_i[Y]$ occurring together in any order; for brevity, it is written as $R_i[X, Y]$, ignoring the curly brackets. Similar notation is followed for the write steps.

A *history* *h* of *T* is a sequence of the steps of *T* representing the execution of the transactions in a possibly interleaved fashion, starting with $W_0[D]$ and ending with $R_f[D]$. A history is *serial* if there is no interleaving, that is, once a transaction starts executing, it finishes without any other transaction executing some step in between. A transaction T_j reads *X* from transaction T_i in a history *h* if $W_i[X]$ is the last write *X* step before $R_j[X]$ in *h*. The *reads-from* relation *rf* of a history *h* is defined as: $rf(h) = \{(T_i, X, T_j) : T_j \text{ reads } X \text{ from } T_i\}$. Two histories *h* and *h'* (of the same set of transactions) are *equivalent* if $rf(h) = rf(h')$. A history *h* is *serializable* if there is a serial history *h'* equivalent to *h*. (This is the *view-serializability* notion [11].)

Example: Consider the history h_1 , for $D = \{X, Y\}$: $W_0[D]R_1[X]R_2[X]R_3[Y]W_2[Y]W_3[Y]W_4[X,Y]R_f[D]$. The reads-from relation $rf(h_1)$ is $\{(T_0, X, T_1), (T_0, X, T_2), (T_0, Y, T_3), (T_4, X, T_f), (T_4, Y, T_f)\}$. The history h_1 has the same reads-from relation as the following serial history h_2 , corresponding to the serial order $(T_0, T_1, T_3, T_2, T_4, T_f)$: $W_0[D]R_1[X]R_3[Y]W_3[Y]R_2[X]W_2[Y]W_4[X,Y]R_f[D]$ and hence is serializable.

A write X step of transaction T_i (also the value that is written) is *useless* in h if no transaction reads this value, that is, there is no T_j for which (T_i, X, T_j) is in $rf(h)$; otherwise, it is *useful*.

The *history graph* of h , denoted $H(h)$, is a directed graph constructed as follows.

The vertex set of $H(h)$ is $T \cup T'$ where $T' = \{T'_{ix} : T_i$ has a useless write X in $h\}$.

The set T' is a set of dummy transactions, one for each useless write in h .

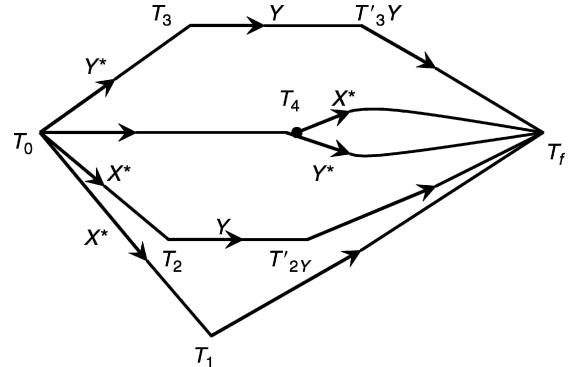
The edge set of $H(h)$ has the following:

- An edge labeled X from T_i to T_j for each (T_i, X, T_j) in $rf(h)$
- An edge labeled X from T_i to T'_{ix} for each useless write X of T_i for all T_i
- An unlabeled edge from each vertex in T' to T_f
- An unlabeled edge from each read-only transaction other than T_f to T_f
- An unlabeled edge from T_0 to each write-only transaction other than T_0

The transaction name itself is used to denote the vertex corresponding to that transaction in the graph. An edge α from T_i to T_j is denoted (T_i, T_j) . Here, T_i is the *positive end* $p\alpha$ of α , and T_j is the *negative end* $n\alpha$ of α . The edge α is also referred to as an *outdirected* edge of T_i and an *indirected* edge of T_j . A *source* is a vertex with no *indirected* edges, and a *sink* is a vertex with no *outdirected* edges. An *X-edge* refers to an edge labeled X . The labeled edges incident to T' are useless; all other labeled edges are useful.

The history graph for h_1 is given in Fig. 1. In the graph, useful edges are indicated by a star (*).

Clearly, two histories h and h' are equivalent iff $H(h) = H(h')$. It is easy to verify that the history graph of a serial history is acyclic. Hence, the history graph of a serializable history will also be acyclic; any topological sort of the vertices will give a serial order of the transactions in an equivalent serial history. (The dummy transactions are ignored.) However, $H(h)$ being acyclic is not sufficient for h to be serializable. For example, the



Serializability. Figure 1. History graph $H(h_1)$.

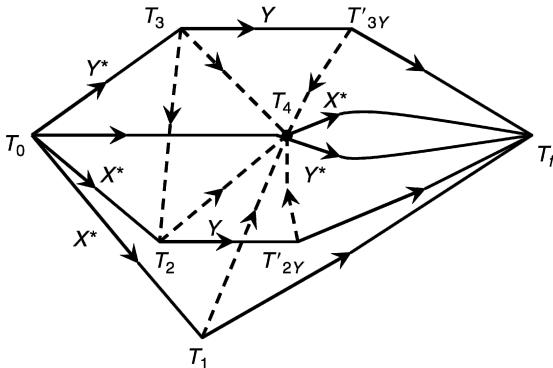
history $W_0[X]R_1[X]R_2[X]W_2[X]W_1[X]R_f[X]$ gives an acyclic history graph, but neither of the two possible topological sorts, (T_0, T_1, T_2, T_f) and (T_0, T_2, T_1, T_f) , gives a serial history which has the same reads-from relation as the original one. Thus for h to be serializable, $H(h)$ must satisfy some other property in addition to being acyclic. That property is characterized in the following, after some more terminology is introduced.

The *coboundary* δS of a subset S of the vertex set of a graph G is the set of edges each with one end in S and the other not in S . By a *coboundary* of G is meant a set of edges that is a coboundary of some subset of the vertex set of G . A *cutset* is a minimal nonnull coboundary. (A set is *minimal* with a given property if it has that property but none of its proper subsets has that property.) An edge α of a coboundary δS is *outdirected* if $p\alpha$ is in S ; it is *indirected* if $n\alpha$ is in S . A coboundary (cutset) δS is *outdirected* if all its edges are outdirected, and similarly it is *indirected* if all its edges are indirected; in either case, it is *directed*. For a data item X in D , a cutset is *X-unsafe* if it is directed and it contains a useful X -edge α and another (useful or useless) X -edge β such that $p\beta \neq p\alpha$. A cutset is *unsafe* if it is *X-unsafe* for some X in D . Any cutset which is not unsafe is *safe*. Note that all undirected cutsets are safe.

A *transaction precedence graph* of h , denoted $TP(h)$, is a graph that contains (i) $H(h)$ and possibly some additional, unlabeled, edges and (ii) no unsafe cutsets.

Theorem: A history h is serializable iff there exists an acyclic $TP(h)$ [12].

A $TP(h)$ can be obtained from $H(h)$ by repeatedly applying the following construction: for each X -unsafe cutset with useful X -edge α and another X -edge β , add an unlabeled edge from $n\alpha$ to $p\beta$, or from $n\beta$ to $p\alpha$, thus making this cutset undirected. Either of these edges is an *exclusion edge*. A transaction precedence



Serializability. Figure 2. Transaction precedence graph $TP(h_1)$.

graph for the history graph in Fig. 1 is shown in Fig. 2. The added exclusion edges are shown in broken lines. For instance, in $H(h_1)$, $\delta\{T_0, T_2\}$ is Y -unsafe: it is outdirected with edges $\{(T_0, T_3), (T_0, T_4), (T_0, T_1), (T_2, T'_{2Y})\}$, it has one useful Y -edge (T_0, T_3) and one useless Y -edge (T_2, T'_{2Y}) . The exclusion edge (T_3, T_2) makes the cutset undirected and therefore safe. The cutset $\delta\{T_0, T_2, T_3\}$ is directed and has two Y -edges, but is safe since both Y -edges are useless.

Note that, for each unsafe cutset, at most one exclusion edge can be added, to get an acyclic $TP(h)$. (Adding both edges will create a directed cycle.) Thus there are two choices for an exclusion edge. If there are n unsafe cutsets, then there are 2^n choices for adding n exclusion edges. With each such choice, the resulting graph can be checked for acyclicity in polynomial time. However, because of the number of choices, checking whether there exists an acyclic $TP(h)$ for a given $H(h)$, in brute-force manner, will take exponential time. It has been shown that a better time is unlikely. That is, checking whether a history is serializable is an NP-complete problem. Hence, serializability under some additional constraints have been studied so as to get polynomial membership test. The constraints limit the serial histories which can be taken as correct executions.

Another Serializability Characterization

First, another characterization of serializability that depicts the nature of the constraints for polynomial membership test is given. It is based on the following property:

In an acyclic $TP(h)$, for any set Ψ of useful X -edges no two of which have the same positive end,

- (a) There is a directed path that contains all the edges of Ψ .
- (b) For each useless X -edge β , there is a directed path that contains β and all the edges of Ψ .

In the following, a transaction that writes X will be called an X -writer. It is *useful* (*useless*) X -writer if the write X step is useful (useless) in h . Note that for some X and Y , the same transaction could be a useless X -writer but a useful Y -writer. The property instrumental to the characterization is the following:

In an acyclic $TP(h)$, for each X in D ,

- (a) There is a directed path from T_0 that contains all the useful X -writers.
- (b) For each useless X -writer T_a , there is a directed path from T_0 that contains all the useful X -writers and T_a .

A *weak order* on X -writers in h is an irreflexive partial order such that

- (a) It totally orders all the useful X -writers and T_0 .
- (b) For any useless X -writer T_a it totally orders all useful X -writers, T_0 and T_a .

A *weak order* in h is the union of weak orders on X -writers in h , one for each X in D .

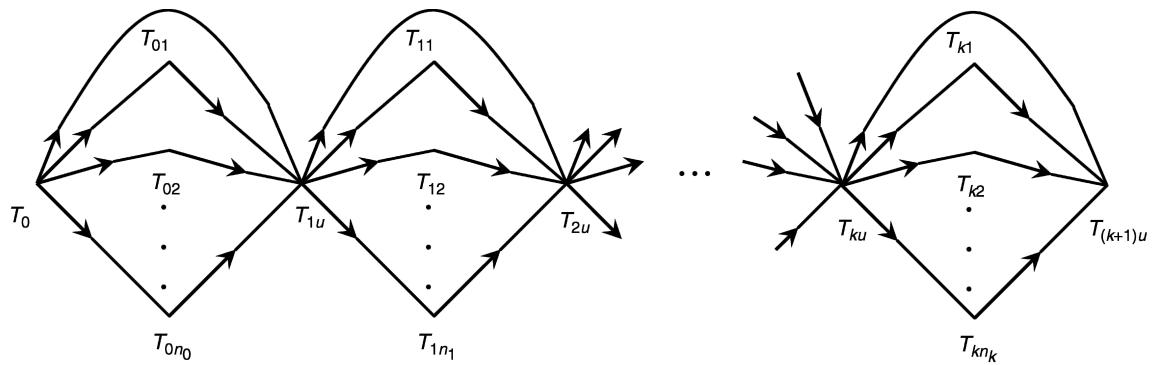
Figure 3 depicts a weak order on X -writers for some X . Here, T_{1w}, T_{2w}, \dots are useful X -writers, T_0 may or may not be useful, and all others are useless X -writers. A weak order q can be added to $TP(h)$ by means of unlabeled edges, one for each element of q . The resulting graph is called $TP[q](h)$.

Theorem. A history h is serializable iff for some weak order q in h there exists an acyclic $TP[q](h)$ [12].

For any weak order q , in each unsafe cutset of $H(h)$ with useful X -edge α and another X -edge β , the weak order edges will introduce a directed path between $p\alpha$ and $p\beta$ and this will induce a directed cycle with one of the exclusion edges; therefore only the other exclusion edge can possibly be added to get an acyclic $TP[q](h)$ graph. Acyclicity of a graph can be checked in polynomial time. Therefore, checking whether there exists an acyclic $TP[q](h)$, for a given q , can be done in polynomial time.

Serializability under Constraints

For read or write steps O and O' , OO' -constraints are defined as: if $O_i[X]$ of transaction T_i occurs before $O'_j[X]$ of transaction T_j in h , for some X in D , then



Serializability. Figure 3. Weak order on X-writers.

T_i must be serialized before T_j . Depending on whether each of these steps is a read (R) or a write (W), the constraints are classified as *WW*-, *WR*-, *RW*- and *RR*-constraints. In addition, *WRW*-constraints are defined as the union of *WR*- and *RW*-constraints. It turns out that *WRW*-constraints impose exactly a weak order q . *WW*-constraints are more stringent than *WRW*-constraints; they impose a total order on all the *X*-writers whereas the latter impose a total order only on useful *X*-writers. Hence there are histories which are serializable under *WRW*-constraints but not under *WW*-constraints. For histories in which there are no useless writes (on any data item), *WRW*-constraints are equivalent to *WW*-constraints. Serializability under *WRW*-constraints, and similarly *WW*-constraints, can be checked in polynomial time. It turns out that *WR*- and *RW*-constraints, individually, do not induce a weak order on h . They leave the choices of exclusion edges for some unsafe cutsets open, and checking membership under each of them is NP-complete.

Serializability under *WW*-constraints corresponds exactly to serializability under the union of *WW*-, *WR*- and *RW*-constraints. It also corresponds exactly to *conflict preserving serializability CPSR* [1], also called *conflict serializability* [15]. Here, the constraints are stated in terms of conflicts: two steps *conflict* if they are on the same data item and at least one of them is a write, and two transactions *conflict* if they have conflicting steps.

A constraint that is not based on conflicts is: if a transaction T_i finishes execution before a transaction T_j starts, in the given history h , then T_i should appear before T_j in an equivalent serial order. The histories

serializable under these constraints are called *strictly serializable* (also *order-preserving serializable* [1,9]).

Different Notions of Serializability

In the serializability definition that the reads-from relation of the given history be the same as that of a serial history, the reads of *all* transactions were considered. This notion has been called *view-serializability* [11]. If only the reads of T_f are considered, then *state-serializability* notion is obtained. Here the final database state is consistent but some transactions may see (read from) an inconsistent database state. A general notion is *S-serializability* where a subset S of T sees a consistent database state. This is characterized as follows:

A step O is *immediately-useful* to another step O' if either O' reads the value (of a data item) written by O , or (i) O is a read of some data item, (ii) O' is a write of another (perhaps different) data item, (iii) both O and O' belong to the same transaction and (iv) O precedes O' in the transaction partial order. The *useful-to* is the transitive closure of *immediately-useful-to*. With respect to a specified subset S of T , a step of T_i is *S-useful* if it is useful to some transaction in S ; otherwise, it is *S-useless*.

The *reads-from relation of h with respect to S* , denoted $rf(h, S)$, is defined as $rf(h, S) = \{(T_i, X, T_j) : T_j \text{ reads } X \text{ from } T_i \text{ and } R_j[X] \text{ is } S\text{-useful}\}$. Two histories h and h' are *S-equivalent* if $rf(h, S) = rf(h', S)$. A history h is *S-serializable* if there is a serial history h' *S-equivalent* to h .

Relative Atomicity

Serializable executions have been considered as correct executions on the premise that a transaction is a unit of

atomicity, and any serial execution is correct. A serial execution is called a *basic* correct execution. This notion of atomicity has been found to be unnecessarily restrictive for many applications. Hence, researchers started refining atomicity units and hence accepting some non-serial histories as basic correct executions. An earliest refinement is in the definition of *nested transactions*, where a transaction consists of several subtransactions, each of which in turn consists of several subtransactions, and so on, with the traditional transactions at the lowest levels of the hierarchy. In one definition of nested transactions [8], each subtransaction is an atomic unit to other subtransactions which are siblings at the same level or their descendants.

The atomicity property in the nested transaction definition has two important characteristics: (i) an atomic unit may consist of *some*, not necessarily all, steps of a transaction; and (ii) some steps may constitute an atomic unit to *some* transactions, not to others. The atomicity property in sagas [5] has characteristic (i). A saga is a two-level nested transaction where each bottom level transaction is an atomic unit for every other transaction. The characteristic (ii) has been called *relative atomicity* [7]. The relative atomicity notion has been generalized in various stages. Garcia-Molina [4] defines *compatible* transactions as a set of transactions whose steps can interleave arbitrarily. If T_i and T_k are not compatible, then the entire transaction T_i is an atomic unit for T_k , and vice versa. If transactions T_i and T_j are compatible, then each step of T_i is an atomic unit for T_j , and vice versa. Then, (i) the steps of T_j can interleave with those of T_i arbitrarily and (ii) *any* number of steps of T_j can be executed after *any* step of T_i . The relative atomicity notion in [3] constrains property (i): T_j is allowed to interleave only at certain points in the execution of T_i , defined as the *breakpoints* of T_i with respect to T_j ; but, whenever T_j is allowed, any number of its steps can be executed. With the *generalized relative serializability* notion of [13], the number of steps of T_j that can be executed at the individual breakpoints is restricted. Further, in the relative serializability notion, the above interleaving restrictions are only with respect to “dependent” steps; non-dependent steps are allowed to interleave anywhere in T_i . The precedence relation among the steps of the same transaction and conflict relations among the steps of different transactions contribute to the dependency. In all these proposals, any execution equivalent to any of the basic correct executions

is considered as a correct execution. Lynch [7] extends the compatibility notion of [4] to subtransactions of nested transactions. Further extensions appear in [12].

Key Application

Serializability has become a de facto yardstick for arguing correctness of concurrent executions of programs. Serializability theory has revealed the amount of concurrency that can be achieved in correct executions and has helped in designing methods of achieving them. The theory has been extended from simple database operations to complex (and even non-electronic) activities, from individual to collaborative applications, and from centralized to highly distributed environments. The theory has also been enriched by adding application semantics along with the syntactic considerations.

Conflict serializability notion has served as a cornerstone to the design of database schedulers. The strength of the notion includes the following two nice properties: (i) it can be checked with a much simpler history graph where vertices correspond to transactions, edges correspond to conflicts, and acyclicity of the graph is a necessary and sufficient condition for the history to be conflict serializable; and (ii) if a history is conflict serializable then the projection of the history over any subset of the transactions is also conflict serializable. The class of histories allowed by the very popular *two-phase locking policy* is a subset of conflict-serializable histories.

Other serializability notions have been used in several advanced applications (for example in design databases [14]).

S

Cross-references

- ACID Properties
- Atomicity
- Concurrency Control – Traditional Approaches
- Multi-Version Serializability and Concurrency Control
- Transaction Management
- Transaction Models – the Read/Write Approach

Recommended Reading

1. Bernstein P.A., Shipman D.W., and Wong W.S. Formal aspects of serializability in database concurrency control. IEEE Trans. Software Eng., SE-5:203–215, 1979.

2. Eswaran K.P., Gray J.N., Lorie R.A., and Traiger I.L. The notion of consistency and predicate locks in a database system. *Commun. ACM*, 19:624–633, 1976.
3. Farrag A.A. and Özsü M.T. Using semantic knowledge of transactions to increase concurrency. *ACM Trans. Database Syst.*, 14(4):503–525, 1989.
4. Garcia-Molina H. Using semantic knowledge for transactions processing in a distributed database. *ACM Trans. Database Syst.*, 8(2):186–213, 1983.
5. Garcia-Molina H. and Salem K. Sagas. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 1987, pp. 249–259.
6. Ibaraki T., Kameda T., and Minoura T. Serializability with constraints. *ACM Trans. Database Syst.*, 12(3):429–452, 1987.
7. Lynch N.A. Multilevel atomicity – a new correctness criterion for database concurrency control. *ACM Trans. Database Syst.*, 8(4):485–502, 1983.
8. Moss T.E.B. Nested Transactions: An Approach to Reliable Distributed Computing. Ph.D. Thesis, Technical Report MIT/LCS/TR-260, MIT Laboratory for Computer Science, Cambridge, Massachusetts, 1981.
9. Papadimitriou C.H. The serializability of concurrent database updates. *J. ACM*, 26:631–653, 1979.
10. Papadimitriou C.H. and Kanellakis P. On concurrency control by multiple versions. *ACM Trans. Database Syst.*, 9:(1):89–99, 1984.
11. Vidyasankar K. Generalized theory of serializability. *Acta Inf.*, 24:105–119, 1987.
12. Vidyasankar K. Unified theory of database serializability. *Fundamenta Inf.*, 14(2):147–183, 1991.
13. Vidyasankar K. Generalized relative serializability. In *Proc. of 9th Int. Conf. on Management of Data*, 1998, pp. 313–327.
14. Vidyasankar K. and Dampney C.N.G. Version consistency and serializability in design databases. In *Proc. 2nd Int. Conf. on Database Theory*, 1988, pp. 368–382.
15. Weikum G. and Vossen G. *Transactional information systems – theory, algorithms, and the practice of concurrency control and recovery*. Morgan Kaufmann, 2002.

Service Bus

- Enterprise Service Bus (ESB)

Service Buses

- Interface Engines in Healthcare

Service Choreography

- Composed services and WS-BPEL

Service Component Architecture (SCA)

ALLEN CHAN

IBM Toronto Software Lab, Markham, ON, Canada

Synonyms

SCA

Definition

The Service Component Architecture (SCA) [1] is a collaborative effort driven by a number of software vendors in the Open SOA (OSOA) [2] Collaboration group to facilitate the building of applications and systems based on service-oriented architecture. The final specification for SCA Version 1.0 was available as of March 21, 2007.

SCA is a set of specifications in the area of service composition, assembly, protocol bindings and policy definitions, where the Service Data Objects (SDO) [3] specification is used to specify how service data can be specified and manipulated. SDO provides a uniform access pattern for heterogeneous data sources, such as XML or relational databases. Although SCA and SDO can work independently of each other, they are often used together to provide a full end-to-end framework for defining SOA applications and systems.

Key Points

Conceptually, SOA is an architecture principle to enable software applications to be exposed as services. However, since SOA itself does not dictate how these services will be packaged or assembled, some of the benefits of SOA such as reuse, manageability and scalability become unpredictable. SCA can speed up SOA adoption by using the SCA Assembly Model to define how services can be declared, implemented and connected to each other.

The basic building block for SCA is an SCA Component, which can be used to declaratively describe the business *services* exposed by an *implementation*, and declare dependency on other business services as *references*. In addition, *bindings* can be applied to any *services* or *references* to describe how a client application can invoke an existing service or how an external service can be accessed, respectively.

SCA also supports a recursive composition model to support the creation of a composite SCA Component. Another aspect of the SCA Assembly Model is the SCA Policy Framework [2], which provides a way to capture the non-functional aspects of an SOA system.

The SCA specification provides a framework for the implementation of scalable and manageable SOA systems, such as the enterprise service bus (ESB).

Cross-references

- ▶ Enterprise Service Bus (ESB)
- ▶ Service Data Objects (SDO)
- ▶ Service Oriented Architecture (SOA)

Recommended Reading

1. Open SOA Collaboration, <http://www.osoa.org/>.
2. SCA Specification, Final Version 1.0, <http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications>.
3. SDO Specification, Final Version 2.1, <http://www.osoa.org/display/Main/Service+Data+Objects+Specifications>.
4. Spring Framework <http://static.springframework.org/spring/docs/2.0.x/reference/index.html>.

Service Composition

- ▶ Composition

Service Item

- ▶ Clinical Order

Service Orchestration

- ▶ Composed services and WS-BPEL

Service Order

- ▶ Clinical Order

Service Oriented Architecture

SERGE MANKOVSKI
CA Labs, CA Inc., Thornhill, ON, Canada

Synonyms

SOA

Definition

Service Oriented Architecture is a conceptual model for integration of software systems where system function is performed by coordinated invocation of services. In this model term service refers to *significant atomic* computational activity that can be invoked over a computer network. Service computational activity is significant in the sense that it is in order of several magnitudes more complex than a function invocation and atomic in the sense that it is a smallest element of functional decomposition in this model.

Historical Background

Historically IT systems are built in a competitive environment where each vendor is trying to develop best possible solutions to their customers and, at the same time, trying to build even more complex systems in attempt to serve any customer need and prevent customers looking for solutions from another vendor. Logic of this process led to development of IT systems that either did not have any means for integration with other systems, or at most had interfaces necessary for integration between within the brand. At the same time companies using IT systems to conduct business were trying to integrate systems in attempt to streamline operations, increase utilization of IT asset and achieve business critical functionality. These competing forces were shaping landscape IT for several decades. Over time there were a number of successful attempts to build large scale integration of IT systems, but more often than not, these integrated solutions themselves were becoming silos in its own right. A notable example of this process was emergence of electronic data interchange (EDI) system that was under development from mid 1960s almost at the same time when DARPA started work leading to development of the Internet. By the time when Tim Berners-Lee developed a first web browser in 1990, EDI was already well established as a successful model of integration of IT functionality across multiple companies.

It is perhaps not possible to pinpoint when exactly SOA way of thinking started. Impact and acceptance of SOA became more evident after successes standardization of technologies that were necessary within the SOA model. Experience and lessons learned by the industry from several decades of development, deployment and operation of various integration architectures, emergence of XML as a universal data interchange format, emergence of new open standards, and tremendous success of Internet have build technological foundation

for SOA adoption. Broad adoption started when critical mass of standards comprising SOA stack has matured. At the same time Open Source Development phenomena made implementations of the standards broadly available. It removed barriers created by proprietary implementations and lowered barrier to entry for new users of the technology. Growth in scope, diversity and rapid pace of change in business requirements and demand business agility created awareness of SOA benefits in business community.

Foundations

SOA as an integration architecture is concerned with all aspects of interactions between IT systems. SOA postulates that a basic element of SOA architecture is service.

SOA system is comprised of a number of *services* deployed over a computer network. *Service* is a basic building element of SOA. Notion of a service is similar to notion of object in object oriented programming, but it is more coarse-grained. For example, a service can represent an important function of an IT system or even entire system all together. Each service has an *interface*. Service interface is *metadata*, or data about data, needed to *invoke* service operation.

In respect to invocation SOA distinguishes two roles – Service Consumer and Service Producer. Service Consumer is a system invoking service and Service Producer performs the service. Service Producer invokes service by message to an instance of Service Producer. Upon receiving an invocation conforming to the service interface, Service Producer executes requested operation using parameters provided by the invocation. Service execution might also include invocation of other services within the SOA system and hence Service Producer can also play role of Service Consumer in respect to another service. When Service Producer completes execution of the operation it replies to Service Consumer with an acknowledgement of successful completion or data containing results of the invocation or indication of fault. This reply from Service Producer to Service Consumer indicates completion of service execution.

It is important to note that service interface is an abstraction that does not take into account details of the network data transport needed to invoke the service or details of service implementation. Use of the interface abstraction allows for definition of service interactions at the higher level of abstraction than in

any other integration architecture. In particular, it allows for definition of service interactions at design time and achieves high degree of flexibility regarding service implementation, location, time, and transport protocol needed for service invocation.

Since interface is separated from service instance it is necessary to associate interface with a service instance in a process that is called *service binding*. SOA allows to perform service binding at any time of SOA system life time. In particular, it can be done at run time by means of *service discovery* and *service lookup*. Service discovery is a process of discovery of services capable to perform a necessary function. Service lookup is a process of finding a service *end point reference* based on a service name, name of a service interface, or the interface metadata itself. End point reference is a piece of metadata that must contain a protocol dependent service address along with optional parameters and session identifier.

Notion of service interface is important from the software engineering point of view because allows for separation of the function performed by service from implementation of service itself. This constitutes a good software engineering practice leading to more robust system design. It also enables very powerful notion of service *orchestration*. Service orchestration is a process of delivering a *composite service* function by coordinated invocation of other services. It is usually done within an orchestration engine that executes the service in accordance with the process definition describing logic and sequence of invocation of orchestrated services along with the necessary *data transformation*. Data transformation is an activity of modifying syntax of the data exchanged between the services to accommodate their interface requirements.

SOA as a conceptual model accommodates wide variations. Any of the architectural concepts highlighted above can be omitted, except concept of service. This is perhaps why it is called service-oriented architecture.

In respect to services there are at least two major types of services:

1. SOAP services use WS-* stack of Web Service Standards. These services are based on a number of international standards developed with W3C and OASIS standardization committees and make extensive use of XML.
2. RESTful services emerged from the world of Open Source. They make use of HTTP protocol and

became “standard-de-facto” for web based services and mush-ups.

Within SOA, there is a wide degree of variation in respect to use of metadata. SOAP services make extensive use of metadata: XML Schema for message syntax, WSDL and Schema for interface definition, WSDL for binding, UDDI for lookup, BPEL for orchestration. RESTful services use HTTP verbs for defining actions and use HTTP for data transport. They do not have formal definition of data syntax beyond URL syntax, no notion of binding, lookup, discovery and orchestration and hence no metadata associated with them.

In respect to invocation there are three types of invocation:

1. *Synchronous* invocation. This form of invocation passes control of execution from Service Consumer to Service Producer and it does not return to Service Consumer until service is completed. This type of invocation is common within both SOAP and RESTful services.
2. *Asynchronous* invocation. In this form of invocation Service Consumer does not need to wait for Service Producer to complete service invocation. Service Consumer can carry on performing its own function, but it requires capability on the part of the Service Consumer to receive message indicating completion of the service. This type of invocation is often done by use of a messaging system. This type of invocation is more common among SOAP services.
3. *Enterprise Services Bus (ESB) based* invocation. In this form of invocation Enterprise Service Bus performs function of discovery, lookup, binding, messaging, data transformation and orchestration. ESB-based invocation can be performed synchronously or asynchronously, but in this case service invocation can be done using abstract Service Producer interface. Service invocation is passed to ESB that performs Service Producer lookup and binding and invocation. If necessary, ESB performs data transformation. If requested service is a composite service, ESB performs necessary orchestration.

Variation in respect to service discovery, lookup and binding range from systems fully bound at design time to systems using run-time semantic-based mechanisms employing artificial intelligence methods and

techniques. Systems performing mission critical function tend to drift towards design time binding. SOA systems aiming to accommodate high rate of changes tend to drift towards sophisticated run-time binding.

There is a wide variation in use and purpose of the SOA systems themselves. WS-* based service architecture tend to be used for enterprise integration. They often use ESB as a back-bone carrying majority of the business related data. It becomes a focal point where enterprise policies can be enforced and formal audit necessary for proving business compliance can be conducted. Because of this highly visible position of SOA systems within enterprise it gives rise to notion of *SOA Governance*. SOA Governance is an on-going activity within an enterprise maximizing leverage of the SOA infrastructure for business purposes. SOA Governance has two distinctive aspects. One aspect refers to ensuring that all aspects of an enterprise functioning within SOA are performing their functions in expected manner by means of enforcing and auditing compliance with business policies, practices. In this aspect SOA provides means for automated support and tractability of decisions and actions performed within enterprise. Automated decision and action support within SOA is achieved by use of service orchestration. SOA Orchestration provides automation of business processes, automatic routing of documents, enforcement of timely document processing, notification of non-compliance, and change management. Tractability of decision and actions is achieved by retaining of service invocation data within the SOA infrastructure. It allows for cross-referencing and correlation of logging data retained with the services and allows reconstructing entire picture of business activity at any point of time. It ensures that at any point of time there is a means of checking if business activities were performed in accordance with law, regulations and in adherence to best practices.

Another meeting of SOA Governance relates to operation of the SOA system itself. Services within SOA have a certain degree of freedom and can change independently from each other as long as interfaces remain unchanged. However they are not completely independent because it is often not possible make them completely independent and there is still some degree of dependence between the services. This requires some level of control over the degree in which services can vary. SOA Governance makes sure that any change within the system does not destabilize or jeopardize

business function performed by the system. It is accomplished by imposing policies to restrict degree of changes in behavior of services, ensure that services continue to perform within established service levels, managing deployment of new services and maintaining operation of the existing services. This from of governance also uses the same automation and tractability infrastructure as the other one.

Key Applications

Enterprise Application Integration, Business Process Optimization.

Future Directions

Deployment of SOA without changing business processes does not produce the same level of return on investment as if deployment is accompanied by changes in the business processes tailored to take advantage of the SOA system. On the other hand changes in business processes trigger changes in the SOA system. In the future, it would be necessary to develop a methodology for SOA deployment. This methodology would have to cover both technical and business aspects as well as provide foundation for understanding of economic impact and, ultimately, quantify return on investment associated with SOA deployment.

Cross-references

- ▶ [BPEL](#)
- ▶ [Enterprise Application Integration](#)
- ▶ [Enterprise Service Bus](#)
- ▶ [Messaging Systems](#)
- ▶ [Mash-up](#)
- ▶ [OASIS](#)
- ▶ [Open Source](#)
- ▶ [RPC](#)
- ▶ [SOAP](#)
- ▶ [UDDI](#)
- ▶ [W3C](#)
- ▶ [Web Services](#)
- ▶ [WSDL](#)

Recommended Reading

1. OASIS Reference Model for Service Oriented Architecture, <http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf>
2. Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Thomas Erl. The Prentice-Hall Service Oriented Computing Series, 2005.

3. Understanding Enterprise SOA. Eric Pulier, Hugh Taylor, Manning, 2005.
4. W3C Web Services Glossary, <http://www.w3.org/TR/ws-gloss/>

Service Request

- ▶ [Clinical Order](#)

Session

SAMEH ELNIKETY
Microsoft Research, Cambridge, UK

Synonyms

[Database interaction](#)

Definition

A database session is sequence of interactions between a client and a database server. The session captures the state of the client's in-flight SQL commands.

Key Points

Session state may contain database objects, such as temporary relations, which are accessible only within the session. For efficiency, some database engines maintain session state per connection rather than per client. In this case, it is called connection state.

A client expects to see the effects of its previous updates to the database. This concept is called session consistency [1] and is illustrated in the following example. A client issues a transaction to buy a book. Then, it sends a subsequent transaction to see the list of ordered books. Session consistency requires the list to contain that book. Session consistency is trivial to implement in a centralized database system, but becomes harder in a distributed database system.

Cross-references

- ▶ [Connection](#)
- ▶ [Strong Consistency Models for Replicated Data](#)

Recommended Reading

1. Daudjee K. and Salem K. Lazy database replication with ordering guarantees. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 424–435.

Set Abstraction

- ▶ Comprehensions

Set-Difference

- ▶ Difference

Shape Descriptors

- ▶ Feature-Based 3D Object Retrieval

Shared-Disk File System

- ▶ SAN File System

Shared-Disk Architecture

PATRICK VALDURIEZ
INRIA, LINA, Nantes, Cedex, France

Definition

In the shared-disk architecture, only the disks are shared by all processors through the interconnection network. The main memory is not shared: each processor has exclusive (non-shared) access to its main memory. Each processor-memory node is under the control of its own copy of the operating system. Since any processor can cache the same disk page, a cache coherency mechanism is necessary.

Key Points

Shared-disk requires a cache coherency mechanism which allows different nodes to cache a consistent disk page. This function is hard to support and requires some form of distributed lock management. The most notable parallel database system which uses shared-disk is Oracle, with an efficient implementation of a distributed lock manager for cache consistency.

Shared-disk has a number of advantages: lower cost, good extensibility, availability, load balancing, and easy migration from centralized systems. The cost of the

interconnection network is significantly less than with shared-memory since standard bus technology may be used between processor nodes. Given that each processor has enough main memory, interference on the shared disk can be minimized. Thus, extensibility can be better, typically up to a hundred processors. Since memory faults can be isolated from other nodes, availability can be very good. Load balancing is relatively easy as a query at any node can access all data on the shared disks. Finally, migrating from a centralized system to shared-disk is relatively straightforward since the data on disk need not be reorganized.

However, shared-disk suffers from complexity and potential performance problems. It requires distributed database system protocols, such as distributed locking and two-phase commit which are complex. Furthermore, maintaining cache consistency can incur high communication overhead among the nodes. Finally, access to a shared-disk is a potential bottleneck.

Cross-references

- ▶ Parallel Data Placement
- ▶ Parallel Query Processing
- ▶ Query Load Balancing in Parallel Database Systems

Shared-Disk Databases

- ▶ Process Structure of a DBMS

Shared-Everything

- ▶ Shared-Memory Architecture

Shared-Everything Databases

- ▶ Process Structure of a DBMS

Shared Health Record

- ▶ Electronic Health Records (EHR)

Shared-Memory Architecture

PATRICK VALDURIEZ

INRIA, LINA, Nantes Cedex, France

Synonyms

[Shared-everything](#)

Definition

In the shared-memory architecture, the entire memory, i.e., main memory and disks, is shared by all processors. A special, fast interconnection network (e.g., a high-speed bus or a cross-bar switch) allows any processor to access any part of the memory in parallel. All processors are under the control of a single operating system which makes it easy to deal with load balancing. It is also very efficient since processors can communicate via the main memory.

Key Points

Shared-memory is the architectural model adopted by recent servers based on symmetric multiprocessors (SMP). It has been used by several parallel database system prototypes and products as it makes DBMS porting easy, using both inter-query and intra-query parallelism.

Shared-memory has two advantages: simplicity and load balancing. Since directory and control information (e.g., lock tables) are shared by all processors, writing database software is not very different than for single-processor computers. In particular, inter-query parallelism is easy. Intra-query parallelism requires some parallelization but remains rather simple.

Load balancing is also easy to achieve since it can be achieved at run-time by allocating each new task to the least busy processor.

However, shared-memory has three problems: cost, limited extensibility and low availability. The main cost is incurred by the interconnection network which requires fairly complex hardware because of the need to link each processor to each memory module or disk. With faster processors, conflicting accesses to the shared-memory increase rapidly and degrade performance. Therefore, extensibility is limited to a few tens of processors, typically up to 16 for the best cost/performance. Finally, since memory is shared by all processors, a memory fault may affect several processors thereby hurting availability. The solution is to use

duplex memory with a redundant interconnect which makes it more costly.

Cross-references

- ▶ [Parallel Data Placement](#)
- ▶ [Parallel Query Processing](#)
- ▶ [Query Load Balancing in Parallel Database Systems](#)

Shared-Nothing Architecture

PATRICK VALDURIEZ

INRIA, LINA, Nantes, Cedex, France

Synonyms

[Distributed architecture](#)

Definition

In the shared-nothing architecture, each node is made of processor, main memory and disk and communicates with other nodes through the interconnection network. Each node is under the control of its own copy of the operating system and thus can be viewed as a local site (with its own database and software) in a distributed database system. Therefore, most solutions designed for distributed databases such as database fragmentation (called partitioning in parallel databases), distributed transaction management and distributed query processing may be reused.

Key Points

As opposed to symmetric multiprocessor (SMP), shared-nothing is often called massively parallel processor (MPP). Many research prototypes and commercial products have adopted the shared-nothing architecture because it has the best scalability. The first major parallel DBMS product was Teradata which could accommodate a thousand processors in its early version in the 1980s. Other major DBMS vendors, except Oracle, have provided shared-nothing implementations.

Shared-nothing has three main advantages: low cost, high extensibility, and high availability.

The cost advantage is better than that of shared-disk which requires a special interconnection network for the disks. By easing the smooth incremental growth of the system by the addition of new nodes, extensibility can be better (in the thousands of nodes). With careful

partitioning of the data on multiple disks, almost linear speedup and linear scale up could be achieved for simple workloads. Finally, by replicating data on multiple nodes, high availability can be also achieved.

However, shared-nothing is much more complex than either shared-memory or shared-disk.

Higher complexity is due to the necessary implementation of distributed database functions for large numbers of nodes, in particular, data placement. Load balancing is more difficult to achieve because it relies on the effectiveness of database partitioning. Unlike shared-memory and shared-disk, load balancing is decided based on data location and not the actual load of the system. Furthermore, the addition of new nodes in the system presumably requires reorganizing the database to deal with the load balancing issues.

Cross-references

- ▶ [Parallel Data Placement](#)
- ▶ [Parallel Query Processing](#)
- ▶ [Query Load Balancing in Parallel Database Systems](#)

Shared-Nothing Databases

- ▶ [Process Structure of a DBMS](#)

Shot Boundary Detection

- ▶ [Video Shot Detection](#)

Shot Segmentation

- ▶ [Video Segmentation](#)

Shotcut Detection

- ▶ [Video Segmentation](#)

SI

- ▶ [Snapshot Isolation](#)

Side-Effect-Free View Updates

YANNIS VELEGRAKIS

University of Trento, Trento, Italy

Definition

A view is an un-instantiated relation. The contents of its instance depend on the view query and the instances of the base tables. For that reason, an update issued on the view cannot be directly applied on the view instance. Instead, it has to be translated into a series of updates on the base tables so that when the view query is applied again on the modified base table instances, the result of the view update command will be observed on the view instance. Unfortunately, it is not always possible to find an update translation such that the change observed on the view instance is the one and only the one specified by the view update command. When this happens for a view update translation, the translation is said to have no *side-effects*. To fully exploit the updateability power of views, it is desired to be able to find update translations that have no side-effects.

Historical Background

Updates on the views were introduced almost simultaneously with views. Their importance has been recognized by Codd himself. In fact, one of the twelve rules that Ed Codd [2] introduced to define what a real relational database is, was referring to the ability of the views to be updateable. In particular, the sixth rule was:

“All views that are theoretically updatable must be updatable by the system”

The term *theoretically updatable* is referring to the ability of finding side-effect-free translations of the view updates.

Foundations

The problem of side-effect-free updates is based on the problem of *Updates through views*. Referring to Figure 1 of that entry, the update translation W of a view update U on a view V is said to have no side-effects if $U(Q_V(I)) = Q_V(W(I))$, where I is the database instance and Q_V is the view query of the view V .

To better realize the problem of side-effect-free view updates, consider a database instance that consists of the 3 tables of Figure 1.

Personnel

Department	Employee
CS	Smith
EE	Smith
Philosophy	Kole

Teaching

Professor	Equipment	Seminar
Smith	Projector	Programming
Smith	Projector	Databases
Smith	Laser	Physics
Kole	Microphone	Databases

Schedule

Course	Room
Programming	10
Databases	10
Databases	23

Side-Effect-Free View Updates. Figure 1. Three base Tables.

Suppose that a view V_1 is defined on top of these three tables through the following view query:

```
select *
from Personnel P, Teaching T, Schedule S
where P.Employee = T.Professor and
      T.Seminar = S.Course
```

The instance of the view will be the relation illustrated in [Figure 2](#).

Consider now an update command on this view that requests the deletion of the tuple $t_d:[EE, Smith, Smith, Projector, Databases, Databases, 10]$. Tuple t_d appears in the view instance due to the join of the 3 tuples $[EE, Smith]$, $[Smith, Projector, Databases]$ and $[Databases, 10]$ of the base tables Personnel, Teaching and Schedule. Deletion of any (or all) of these tuples will achieve the desired result of deleting tuple t_d from the view. However, any such deletion will have additional effects in the view instance. For instance, the

removal of tuple $[EE, Smith]$ from Personnel will also eliminate the view tuples that are immediate before and after t_d . Similar observations can be made for the tuples in the other two base relations. In fact, for the particular update, it can be shown that there is no change that can be made on the base tables to achieve the desired tuple deletion without any additional changes, i.e., side-effects, in the view instance.

Side-effects are not observed only on deletions but also on insertions. For instance, consider the update command that requests the insertion of tuple $t_i:[Economy, Smith, Smith, Projector, Databases, Databases, 10]$ in V_1 . For the appearance of t_i in the view V_1 to be justified, tuples $[Economy, Smith]$, $[Smith, Projector, Databases]$ and $[Databases, 10]$ need to exist in the instances of Personnel, Teaching, and Schedule, respectively. The last two are already there, but not the first. The translation of the insert command on the base tables will insert $[Economy, Smith]$ in Personnel. Unfortunately, due to the value “Smith” in its attribute *Employee*, tuple $[Economy, Smith]$ will be able to join with every other tuple of table Teaching that has value “Smith” in the attribute *Professor*. This will introduce additional tuples in the V_1 instance that the insert command did not request.

Base tables, i.e., tables that have been defined through the “create table” command, have standalone instances, thus, update commands on them can be implemented without side-effects by simply modifying their materialized instance accordingly. If views are to be used as any other table, a view needs to show the same behavior as base tables. This means that update commands need to have translations that generate no side-effects in the view instance. It would have been really surprising for an application or a user that is not aware that a relation she is interacting is actually a view, to request the deletion (or insertion) of a tuple and then see additional tuples disappearing from the view (or appearing in it).

To cope with the view update translation side-effects one option is to leave the burden to the database administrator who defines the view. During the view definition, the administrator is responsible to specify not only the view query but also how exactly each update is translated to updates on the base tables [7] and make sure that side-effects will not occur. The drawback of this option is that it requires a lot of knowledge and experience from the administrators. If the administrator determines that for a

V_1

Department	Employee	Professor	Equipment	Seminar	Course	Room
CS	Smith	Smith	Projector	Programming	Programming	10
CS	Smith	Smith	Projector	Databases	Databases	10
CS	Smith	Smith	Projector	Databases	Databases	23
EE	Smith	Smith	Projector	Programming	Programming	10
EE	Smith	Smith	Projector	Databases	Databases	10
EE	Smith	Smith	Projector	Databases	Databases	23
Philosophy	Kole	Kole	Microphone	Databases	Databases	10
Philosophy	Kole	Kole	Microphone	Databases	Databases	23

Side-Effect-Free View Updates. Figure 2. The view V_1 instance.

given view update there is no translation that has no side-effects, she can make the view not to accept this kind of updates, or allow the side-effects to happen if she believed that this is the semantically correct behavior.

Instead of letting the administrator deciding whether an update should be allowed or not, an alternative solution is to develop methods to perform this test automatically. Based on this idea, Keller [4] developed five criteria to characterize the correctness of a view update translation. The first of these criteria requires the translation to have no side-effects. A consequence of this is that keys of the base relations have to appear in the views, i.e., cannot be projected out. This reduces the cases in which side-effects may appear in the views, but does not completely eliminate them. Keller studied the different choices that exist when translating updates on select, project, select-project and select-project-join views, and provided algorithms for update translation for each case. These algorithms are guaranteed to respect his 5 criteria. For a given update on the view, there may be more than one algorithms that can be used, i.e., more than one possible translations. Which one to be used is a decision that is provided by the database administrator at the moment of view definition [5].

Dayal and Bernstein [3] introduced the notion of the *view-trace* and the *view-dependency* graph. They are graphs that model the dependencies between the attributes of the base tables as determined by the schema

and the view definitions. Through them one can determine whether there is an update translation that has no side-effects. For each update on the view, either a unique side-effect-free translation is found and is applied, or the update is not allowed to occur. This approach eliminates the need of an administrator involvement, but cannot be applied in cases in which side-effect generated translations are allowed to occur if they are semantically meaningful.

A different method to determine the kind of updates a view can accept without generating side-effects is through the *constant complement*. Two views are considered complementary if given the state of each view there is a unique corresponding database state. This means that when the instance of one of these views changes (due to an update) while the instance of the other is kept constant, then there is a unique database instance from which the instances of the two views are generated. In other words, the correct translation of the view update is unique [1]. Unfortunately, given a view V , finding its view complement has been shown to be NP-complete even for views with very simple view definition queries.

In summary, it is not always guaranteed that there is a unique side-effect-free translation of a view update. In practical situations, updates are typically allowed on the views. If there are more than one translations of a view update, some criterion may be used to select one of these translations, e.g., the minimality of the changes in the database instance, or

some specific parameters that the data administrator specified at the time of view definition. Another approach is to disallow updates on the view that have more than one translation. In the presence of side-effect generating translations, one approach is to allow the translation to take place, allowing that way the side-effects to appear on the view, or to disallow the update completely.

But what would happen in cases in which an update on the view is absolutely necessary, as is the case of an application that can access the database only through a view interface without being aware of the fact that the relation it is accessing is actually a view and with the need to perform updates on it as it would have done if it was a base table? An idea proposed by Kotidis et al. [6] is the following. When an update command is issued on the view, the change in the view instance must be exactly the one described by the update. However, any change on the base table should not take place unless it is implied by the semantics of the view query and the update command. For instance, the deletion of the tuple [EE, Smith] from Personnel as a translation of the delete command for the view tuple [EE, Smith, Smith, Projector, Databases, Databases, 10] mentioned above, would have implied that the reason that the view tuple is deleted is that Smith stopped being affiliated with the EE department. However, neither the semantics of the update command, nor the semantics of the view query imply something like that. Similar claims can be done for tuples [Smith, Projector, Databases] and [Databases, 10] of tables Teaching and Schedule, respectively. For the specific view tuple deletion, the claim is that no change should be observed in the instances of the three base tables, but tuple [EE, Smith, Smith, Projector, Databases, Databases, 10] will be removed from the view instance. This behavior will only be possible if one can accept views whose instances are not exclusively determined by the results of their view queries on the base tables, but also from the update commands that have been issued on them.

Key Applications

Achieving side-effect-free updates on the views is of great importance for systems that provide access to their data through views, but at the same time need to hide from their users or the applications that use the system the fact that they are dealing with views and not actual relations.

Cross-references

- ▶ Provenance
- ▶ Updates through Views

Recommended Reading

1. Bancilhon F.B. and Spryatos N. Update Semantics of Relational Views. *ACM Trans. Database Syst.*, 6(4):557–575, 1981.
2. Codd E.F. Is Your DBMS Really Relational? *Computer-World*, 1985.
3. Dayal U. and Bernstein P. On the correct translation of update operations on relational views. *ACM Trans. Database Syst.*, 8(3):381–416, 1982.
4. Keller A.M. Algorithms for translating view updates to database updates for views involving selections, projections, and joins. In Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1985, pp. 154–163
5. Keller A.M. Choosing a view update translator by dialog at view definition time. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 467–474.
6. Kotidis Y., Srivastava D., and Velegrakis Y. Updates through views: a new hope. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
7. Rowe L.A. and Shoens K.A. Data abstractions, views and updates in Rigel. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 71–81.

Sight

- ▶ Visual Perception

Signal Transduction Networks

- ▶ Biological Networks

Signature Files

MARIO A. NASCIMENTO

University of Alberta, Edmonton, AB, Canada

Definition

A signature file allows fast search for text data. It is typically a very compact data structure that aims at minimizing disk access at query time. Query processing is performed in two stages: filtering, where false-negatives are guaranteed to not occur but false-positives may occur and, query refinement, where false-positives are removed.

Historical Background

Efficient and effective text indexing is a well-known and long-standing problem in information retrieval. While inverted files are nowadays a *de facto* standard for text indexing, in the early days, its storage overhead was not acceptable for larger datasets. In addition, accessing an inverted file on disk would require a relatively large number of (expensive) disk seeks. The main motivation for signature files is to allow fast filtering of text using a linear scan of the signature file for finding text segments that may contain the queried term(s). Given that the found segments may be false-positives, a refinement step is required before the final correct answer is returned. The main compromise in signature files lies in how to build signatures for terms and for text segments that allow low storage overhead, fast disk access, and minimizes the ratio of false-positives.

Foundations

Let T be a text to be indexed that is divided into non-overlapping blocks T_i each containing b contiguous terms. For each block T_i a binary signature $S(T_i)$ is built. Assume a hashing function $h(\cdot)$ that takes as an argument a term and returns a signature of length B . The signature for a block T_i can then be obtained by performing a bitwise-OR of the signatures of the terms in that block typically excluding the terms in the stop-list. The signature file for T is then the set of signatures for its T_i . A query term Q is also mapped into a signature $h(Q)$. At query time each block signature $S(T_i)$ is compared to the query's signature $h(Q)$. Denoting the bitwise-AND operator by “ $\&$ ” one can show that if $S(T_i) \& h(Q) = h(Q)$ then block T_i may contain the query term Q and is considered a candidate answer. However, in order to guarantee that only correct answers are returned, all candidate answers must be refined to ensure that they do contain query term Q . The reason for such false-candidates is that when a block signature is built, a particular bitstring matching the query's signature may appear as an incidental combination of different signatures.

Consider the sample text T : “To be, or not to be: that is the question” (punctuation marks can be ignored without loss of generality). Assume further that a hashing function $h(\cdot)$ is such that it hashes all the terms in this text as shown in [Table 1](#).

The resulting block signatures will depend on the value chosen for b . [Table 2](#) shows the resulting blocks T_i and their respective block signatures $S(T_i)$ assuming $b = 2$ and the hashing signatures shown in [Table 1](#).

If Q = “question,” then T_5 satisfies the criterion $S(T_5) \& h(Q) = h(Q)$. In this particular case, T_5 indeed contains Q and the text is selected as a true-positive answer. Consider now the case where Q = “to.” In this case $(S(T_1) \& h(Q)) = (S(T_3) \& h(Q)) = (S(T_5) \& h(Q)) = h(Q)$. In query refinement the actual blocks T_1 , T_3 and T_5 need to be read from disk and inspected. While T_1 and T_3 do contain query term Q thus being true-positives, block T_5 does not, i.e., it is a false-positive. This illustrates the major drawback of signature files. While one can safely discard a block T_i if $S(T_i) \& h(Q) = h(Q)$, the same is not true otherwise. Without the query refinement step the query's answer is prone to contain false-positives, which is typically not acceptable. Several factors need to be considered when aiming at minimizing the probability of false-positives. For instance, the length of the produced signatures (B), the number of terms per text block (b) and the number of bits randomly set in the signatures (which is denoted by n). For instance, for an optimal selection of $n = B \ln(2)/b$ then the false-positive probability is $1/2^n$ [1].

There are other important issues to be considered in addition to minimizing the false-positive probability, and thus the overhead of query refinement, when using signature files. In principle the query signature could be also be the result of a bitwise-OR between several terms, hence in principle leading to the possibility of querying for phrases. Unfortunately, phrase queries are not well supported by signature files. This is due to the fact that the bitwise-OR operation used to produce the block and query signatures does not preserve any notion of order among the terms

Signature Files. Table 1. Sample term signatures

Term	To	Be	or	not	that	is	the	question
H(term)	100100	011000	010010	101000	001100	010001	100001	000110

in the block. Therefore, two blocks with very different term ordering will have the exactly same block signature, which is an obvious problem. Another related problem occurs when terms in a phrase query occur in the boundaries of blocks. For instance, consider if one searches for $Q = \text{"not to,"}$ which would yield $h(Q) = 101100$. Using the block signature illustrated in [Table 2](#), one would find that while T_1 and T_2 would be candidate blocks all would be considered false-positives in the query refinement and one would be unable to find that the queried phrase is indeed in T . It would be missed altogether for being in the boundaries of the constructed text blocks.

Although less popular, negation queries, where one searches for a document not containing a query term, can also be answered using signature files. Recall that in the discussion above if $S(T_i)$ had at least the same bits set as $h(Q)$ then T_i could be answer, pending further checking. If a block T_i does not have at least the same bits as $h(Q)$ set is definitely an answer to the negation query. However, if it has those same bits set, it may still be an answer because those bits may have independently been set by other terms. Thus, in this case further checking is required as well. As an illustration consider the term $Q = \text{"writer"}$ and assume that $h(Q) = 100010$. Since $S(T_i) \& h(Q) \neq h(Q)$ for $i = 1, 3$ and 4 one can be sure that those blocks do not contain Q and therefore, up to that point, T could be an answer to query Q . However the fact that $S(T_i) \& h(Q) = h(Q)$ for $i = 2$ and 5 does not necessarily mean that those blocks contain Q . In fact, only upon checking the text of the blocks one could verify that they do not and consequently classify the text as satisfying the query.

It should be clear by now that the crucial issue in dealing with signature files is to minimize the overhead of query refinement. The more blocks needed to be further check the closer the performance will be to that

Signature Files. [Table 2](#). Block signatures for “To be, or not to be: that is the question”

Block # (i)	Terms in the block (T_i)	Block Signature $S(T_i)$
1	To be	111100
2	or not	111010
3	to be	111100
4	that is	011101
5	the question	100111

of reading the whole dataset, which is obviously not desirable. This issue can be addressed by using lengthier signatures possibly through more complex hashing schemes. Due care is needed though as excessively long signatures detract from the claimed low storage overhead yielded by the signature files.

Despite its limitations above, signature files offer the possibility of very efficient search on disk. Given that the only operation necessary is to read and compare binary signatures in a linear fashion, no relatively expensive disk seeks are necessary during the file scan. Furthermore, the inspection of the signature blocks can be implemented very efficiently in memory. Updating the indexed texts can also be carried out with low overhead in a fairly straightforward way.

Bit-Slice Signature File

Even though the framework presented above is effective, fairly efficient and relatively simple to implement it can be further optimized by exploring the layout of the signature file. The typical layout is to have all block signatures written contiguously in a file. At query time, each and every block is read from the disk and compared to the query signature, regardless of which bits are set in the same. The bit-slice signature aims at reducing disk access per query by exploiting the fact that likely few bits are set in the query signature and in the text blocks.

In order to illustrate the idea, take [Table 2](#) and transpose it, i.e., instead of having one row per signature, one will have one row per bit, i.e., the j th row will contain the j th bit of every block signature, ordered from most to less significant. [Table 3](#) shows the resulting table after “transposing” [Table 2](#).

Notice that the idea when checking whether $S(T_i) \& h(Q) = h(Q)$ is to look for common set bits in corresponding positions. For instance, when searching for a block that contains $Q = \text{"or"}$ one needs to inspect the second and fifth bits since $h(Q) = 010010$. Using the bitslice S_2 one can infer that blocks T_1, T_2, T_3 and T_4 have the 2nd bit set and therefore they could contain Q if they also have the 5th bit set. Similarly using S_5 one can infer that blocks T_2, T_4 and T_5 could contain Q is their 2nd bit is also set. If $S_2 \& S_5$ is computed, one obtains the index to the blocks that need be verified. Interestingly enough, this is equivalent to performing the intersection of the two sets of terms obtained by inspecting each bit individually. In this case $S_2 \& S_5 = 01010$ and hence blocks T_2 and T_4 are candidate

blocks. Note that as before, candidate blocks need to be further refined as false-positives are still possible.

While the query processing using plain signature files would require a linear scan of the whole signature, using bitslice signature this is typically not the case. The tradeoff to be considered is that now one needs to perform (relatively expensive) random disk access in both the bitslice file and the signature files. As well, the storage overhead for the bitslice file, nearly as large as the signature file itself needs to be taken into account. Just like the more straightforward case, a number of parameters need to be considered to produce efficient bitslice signature files.

Signature Trees

As an alternative to simple flat signature files one can also arrange the signatures in a hierarchical balanced tree structure, similar to a B^+ -tree. Once again, the idea

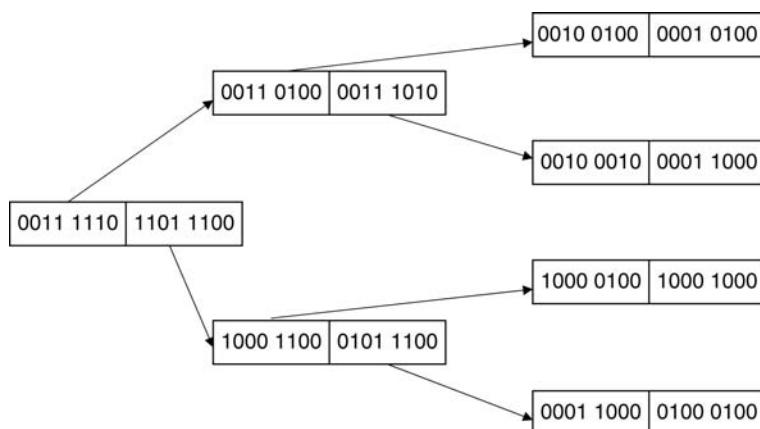
is to trade the few disk seeks required by a linear scan of the signature file by additional seeks that would allow pruning, hence not reading large portions of the signatures.

Assume that a given text has already been broken into blocks T_i and their signatures $S(T_i)$ obtained as discussed above. A signature-tree (or S-tree as it is called in [4]) can be constructed as follows. A set of block signatures are clustered together and stored in a disk page. The number of signatures per page depends primarily on the signature size and the page size. Each entry in such a node is a signature $S(T_i)$ and points to the actual text block T_i , which, as before will be needed to check for false-positives. Note that at this point it would be trivial to process a query by simply traversing all constructed nodes, which would function as a regular signature file. Fortunately, it is possible to avoid reading many of these blocks if a tree structure is used on top of the block signature nodes.

A tree can be constructed by creating an upper layer of nodes that will point to the first layer (which will become leaves of the tree). Each entry in a non-leaf node points to a leaf node containing entries for signatures $S(T_i)$, $S(T_j)$, ..., $S(T_k)$, thus its entry signature will be the bitstring $(S(T_i) \mid S(T_j) \mid \dots \mid S(T_k))$, where “ \mid ” denotes the bitwise-OR operator. The same reasoning can be applied recursively replacing the leaf nodes with the nodes of the current upper level. The sample tree depicted in Fig. 1, assuming an eight bit block signature and a disk page that can fit two signatures, illustrates the result of this process using the set

Signature Files. Table 3. Bitslices for the block signatures in Table 2

Bit # (i)	Bitslice (S_i)
1	11101
2	11110
3	11110
4	10111
5	01011
6	00011



Signature Files. Figure 1. A sample signature-tree. The actual block signatures are at leaf level and point to the respective text block (not shown for simplicity).

of signatures at the leaf level. For instance, the entry 0011 1110 in the root node points to a node containing signatures 0011 0100 and 0011 1010 (indeed $0011\ 0100 \mid 0011\ 1010 = 0011\ 1110$). Similarly the entry 0011 0100 points to the leaf node containing signatures 0010 0100 and 0001 0100.

An important issue is how to cluster signatures in the leaf nodes and, similarly how to group nodes under a single entry in the upper levels of the tree and so on and so forth until the root node. Since one of the possible criteria for this clustering task is tightly related to how the query is processed, the latter is discussed first.

Query processing starts by traversing the signature tree down from the root node choosing which subtree to traverse based on the probability that the subtree contains a candidate block. Assume that a node N has m (signature) entries N_i and it is pointed to by a parent node P under an entry with signature $P_i = (N_1 \mid N_2 \mid \dots \mid N_m)$. The query starts with P_i being each of the root entries. If $P_i \& h(Q) \neq h(Q)$ then it is certain that the subtree pointed by that entry cannot contain a candidate block and it is discarded. Given the way the tree is constructed the bits set are propagated from the leaf nodes up to the root entries, therefore if a given block in that subtree contained $h(Q)$ the root of that subtree entry would necessarily contain $h(Q)$. All is needed now is to repeat the same reasoning using as the new root the node pointed to by the candidate root entry. Note that all candidate subtrees need be traversed and, when finally reaching the leaf level, query refinement step is still required.

As an illustration consider the signature tree in Fig. 1 and assume $h(Q) = 1000\ 0100$. Starting from the entries in the root node one can discard the upper subtree in the Figure since $0011\ 1110 \& h(Q) \neq h(Q)$. The lower subtree needs to be traversed given that $1101\ 1100 \& h(Q) \neq h(Q)$. However, at that point and the expense of one single disk access, half of the signatures can be safely discarded, illustrating the pruning power of signature trees. Of the two subtrees only the one pointed by first entry (1000 1100) needs to be read. Of the two block signatures found in the corresponding leaf nodes, only the one corresponding to signature 1000 0100 needs to be retrieved for the mandatory refinement step.

Given the query processing reasoning above if too many bits are set per block signature, the entries in the non-leaf nodes will quickly have too many bits set and

therefore be unable to help pruning the traversal of the tree. If too many subtrees are traversed, the savings of not reading all signatures is bound to be offset by the additional disk seeks. Clearly the less bits are set higher up in the tree the more selective the traversal will be. This provides a criterion for cluster signatures within a node. Let $W(S_i)$ be defined as a function that returns the number of bits set in signature S_i . Clearly, for any pair of signatures S_i and S_j , $W(S_i \mid S_j) \geq W(S_i) + W(S_j)$. Thus the driving criteria for clustering signatures together is to minimize the value of $W(\cdot)$ over the bitwise-OR'ed signatures. Just as in the case of plain signature files and bitslice signature files, a number of parameters have to be set in order to obtain efficient signature trees.

Key Applications

Text indexing and search.

Cross-references

- ▶ [B+-Tree](#)
- ▶ [Disk](#)
- ▶ [File Systems](#)
- ▶ [Hash Functions](#)
- ▶ [Inverted Files](#)
- ▶ [Text Indexing and Retrieval](#)

Recommended Reading

1. Baeza-Yates R.A and Ribeiro-Neto B.A. Modern information retrieval. ACM Press/Addison-Wesley, 1999.
2. Christos F. Access methods for text. *ACM Comput. Surv.*, 17(1):49–74, 1985.
3. Justin Z., Alistair M., and Kotagiri R. Inverted files versus signature files for text indexing. *ACM Trans. Database Syst.*, 23(4):453–490, 1998.
4. Uwe D. S-tree: a dynamic balanced signature index for office retrieval. In Proc. 9th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1986, pp. 77–87.
5. William B.F. and Baeza-Yates R.A. Information Retrieval: Data Structures & Algorithms. Prentice-Hall, 1992.
6. Witten I.H, Alistair M., and Bell T.C. Managing Gigabytes: Compressing and Indexing Documents and Images (2nd ed.). Morgan Kaufman, 1999.

Signatures

- ▶ [Digital Signatures](#)

Similarity and Ranking Operations

MICHAEL HUGGETT

University of British Columbia, Vancouver,
BC, Canada

Synonyms

Association; Correlation; Matching; Proximity; Ordering; Relevance

Definition

Similarity and ranking operations are fundamental to *information searching*, in which a user generates a *query* phrase of one or more words that reflects an *information need*. The query is used to find related items that satisfy that need.

Similarity operations quantify the *resemblance* or *alikeness* between two information objects. An *information object* is a conceptual unit, most typically described as a *document*, but also taking the form of a term, phrase, paragraph, page, section, chapter, article, book, or script, etc. Information objects may be printed or digital.

Similarity judgments may be subjective (performed by a user) or algorithmic (performed by a computer). Depending on the method of evaluation, likeness judgments may be semantic (i.e., within the meaning of a document), structural (i.e., within parts of speech, position in a document, or pattern of links between documents), or statistical (i.e., within correlations between document attributes). Statistical methods that model the distribution of terms in a corpus are most common in information retrieval (IR).

In typical IR systems, documents are preprocessed to extract representative *keyword* terms. The terms of a user's query are then compared with the keywords of each document to find the best matches. Given a document that satisfies an information need, its keywords may themselves be used to query for related documents.

Ranking operations are based on the numeric result of a similarity measure: given a query representing an information need, the documents that are scored as most similar to the query are most highly ranked. Rank is typically presented to the user as a list of document descriptors (esp. titles) sorted monotonically in decreasing order of similarity score.

Historical Background

The history of information retrieval is parallel to and largely separate from that of database research, since the goal of IR is to find semantically related information in an arbitrary corpus of unstructured documents, whereas the traditional relational database model searches within specified value ranges in pre-defined fields. Information retrieval produces "best guess" matches for a given keyword query, whereas relational databases return items that are *true* of a (typically boolean) query statement. Information retrieval is uncertain and probabilistic; its similarity and ranking operations are vital to this distinction.

Notions of similarity and ranking are fundamental to every-day information needs, as intelligent agents (e.g., people, animals) define target objects (e.g., food) that meet certain goals (e.g., survival), and then seek objects in the environment that meet these goals. As such, the topics of similarity and ranking have been well-discussed by cognitive scientists who study the structure of knowledge (e.g., Tversky, 1975), and before them by philosophers from the Enlightenment (e.g., Hume) back to Ancient Greece (e.g., Aristotle).

By the start of the Information Age in the mid-20th century, discrete mathematics was already the basis of information retrieval research. Proposed techniques included the comparison of attribute lists between documents, called *association* (Yule, 1912); ranking as a process of ordinal measurement (Stevens, 1946); *distance metrics* that quantify the differences between strings, originally used for error detection (Hamming, 1950); the automatic indexing of documents based on the statistical distribution of their terms (Luhn, 1957); and the weighting of terms to improve retrieval accuracy (Maron and Kuhns, 1960).

Beginning in the 1960s, these techniques were refined and combined into the *associative* and *probabilistic* approaches that form the basis of current practice—although earlier explorations can claim to have produced the first operational information retrieval system (e.g., Goldberg's *Statistical Machine* as the first electronic system, c. 1927, and Tillett's *QUEASY* as the first system on a general-purpose computer, c. 1953).

The associative approach was consolidated and evaluated exhaustively in the first viable product of modern information retrieval: the SMART system (Salton, 1966), which continues to be much imitated. The probabilistic approach thereafter established a solid theoretical foundation that had been lacking in

earlier work, and over a series of refinements improved retrieval accuracy to become a standard benchmark (Robertson & Sparck, 1976). Some researchers believe the probabilistic approach represents the future of information retrieval.

Foundations

Similarity refers primarily to alikeness based on shared attributes; it is sometimes described as *association*, *relatedness* or *relevance*. Computational similarity operates by comparing lists of representative terms derived from information objects. Comparisons are either between a query and document, or between two documents. In the latter case, if a user has an exemplar document in hand that represents an information need, that exemplar may be used to “give me more like this one”.

Automatic similarity and ranking operations require information objects in digital form; paper documents may be digitized using optical character recognition (OCR). Similarity comparisons are necessarily automatic in systems that index and organize large corpora of documents. Some systems anticipate significant human interaction, but may not be practical since human similarity valuations are slower, and inter-evaluator consistency is poor.

Information retrieval systems typically encode documents as *term vectors*. Each document in a corpus is assigned a term vector. Words in the document are typically *stemmed* to pool together related words such as *stemmer*, *stemmed*, *stemming*, etc. under a single unique root term, e.g., *stem*. Each unique term is assigned a specific cell in the term vector that contains a numerical value. *Binary vectors* show a ‘1’ in a cell if the document contains the term, ‘0’ otherwise. *Weighted vectors* use either integer values to record the number of times each term appears in the document, or real-number values that express the degree to which the term is representative of the document. All the term vectors in the same corpus use the same (binary or weighted) numbering scheme.

All the term vectors in the same corpus are configured the same way. Two types of configuration are typical. The first type reserves a cell for each unique term that appears in the corpus, and numerical values are assigned only to cells for terms that appear in the document. In a large corpus, document vectors can contain many zeroes, and thus be inefficiently *sparse*.

The second type of configuration involves the use of document *keywords*: a small set of discriminating terms from within the document that represent its content. Keywords are fundamental to *indexing*, whose goal is to identify terms that best identify a document with respect to other documents in a corpus. Using a term-weighting algorithm, a score is calculated for each term in each document, and each document is assumed to be best described by its highest-scoring terms. These terms are chosen as the document’s keywords and low-scoring terms are ignored.

The use of such automatically-extracted keywords provides some advantages. Similarity scores can be calculated more efficiently with a small number of representative keywords than by using all the terms in a document. Keywords also provide a human-readable summary of document content without requiring that all documents be read and evaluated manually. Keywords require less space for term storage: a document term vector is a compact mapping from each keyword to its numerical value. Weighted keywords are standard in IR; more complicated methods than this seldom justify the additional complexity and difficulty (Cleverdon, Mills, & Keen 1966).

Ranking refers to any function that follows the *Probability Ranking Principle*, which states that IR systems are most effective when the documents that they retrieve in response to a query are ordered in decreasing probability of relevance to the user’s information need. Ranking is always performed as a result of some user-generated query.

Associative Similarity

In associative similarity, document term vectors define a metric space: a corpus with n unique keywords can be described as a Euclidean n -space, in which each document vector describes a point, line, or hyperplane. Together, the document vectors of a corpus comprise a matrix subject to algebraic methods for space division and summarization. Associative similarity is also commonly known as the *vector space model*.

The associative approach finds documents most similar to a query by matching the terms of the query vector against all of the document term vectors in the corpus. The matching process typically uses an index to retrieve candidate document vectors. The documents that generate the highest similarity scores are assumed to be those that best meet

the user's information need as represented by the query. Documents are then presented to the user in a ranked list sorted in decreasing order of similarity score.

There are several associative vector methods that can be applied to binary and weighted document vectors to generate a similarity score. The simplest method counts the number of found terms in common between two binary vectors: this produces a *simple matching coefficient*, which is useful under the assumption that *any* matching is important, for example when two objects are assumed to be incomplete descriptions of each other. In practice, this assumption proves to be coarse and inaccurate, and most similarity operations normalize the simple matching coefficient by the lengths of the input vectors. For weighted vectors, the inner product of two vectors is used as the equivalent of the simple matching coefficient, and sum-squares of the vectors are used for normalization. **Table 1** shows binary and weighted versions of four common similarity measures. The weighted cosine method is especially popular, and represents a geometrically accurate interpretation of correlation.

In practice, these methods are virtually interchangeable, since all their scores increase monotonically given the same input data. However, it is notable that for a given method and query vector, multiple target documents with significantly different term vector compositions may generate the same similarity score. Furthermore, the effective domain of similarity scores in n -dimensional space can vary dramatically between different similarity functions. Such results suggest that different similarity operations may be more or less appropriate for corpora with different types of term distribution (Jones & Furnas, 1987) [2].

Probabilistic Retrieval

Probabilistic retrieval is a decision-theoretic process based on the idea that for each query, documents should be retrieved if they maximize the probability that they are *relevant* to the query, $P(\text{relevance}|\text{document})$. Whereas associative similarity measures are largely ad hoc, probabilistic retrieval is based on an explicit theoretical framework. The intuition is that users will find documents relevant if the documents have a certain distribution of attributes, and that the probability of relevance depends on how closely a document's distribution of attributes matches the distribution sought by the user.

Since the probabilistic model depends on the presence or absence of terms, documents are represented with binary term vectors, writing a '1' in each cell that a given term is present, '0' otherwise. In its simplest form, the model assumes that errors of false-positive choices (deemed relevant when irrelevant) and false-negative choices (deemed irrelevant when actually relevant) are negligible.

For a given document represented by term vector X , probability of relevance $P(R)$ and probability of irrelevance $P(\bar{R})$, the document is judged relevant if $P(R|X) > P(\bar{R}|X)$, i.e., if the following discriminant function $g(X)$ is greater than 1:

$$g(X) = \frac{P(R|X)}{P(\bar{R}|X)}$$

Using Bayes' Law, this is rewritten as:

$$g(X) = \frac{P(X|R)P(R)}{P(X|\bar{R})P(\bar{R})} \approx \frac{P(X|R)}{P(X|\bar{R})}$$

since $P(R)$ and $P(\bar{R})$ are constant for each document. The calculation of $P(X|R)$ and $P(X|\bar{R})$ depend on the probabilities of each term in the document. For each term, the following table is used, with N the number of documents in the corpus, R the number of documents relevant to the query, n the number of documents that contain the term, and r the number of relevant documents that contain the term:

	Relevant	Non-relevant	
documents w/ term	r	$n - r$	n
documents w/o term	$R - r$	$N - n - (R - r)$	$N - n$
	R	$N - R$	N

Thus for each binary term x_i in X that matches a term in the query, the relevance function is re-written as a ratio of relevant to non-relevant portions:

$$\begin{aligned} g(X) &\approx \frac{P(X|R)}{P(X|\bar{R})} = \frac{\prod_i P(x_i|R)}{\prod_i P(x_i|\bar{R})} \\ &= \sum_i x_i \log \frac{r_i/(R - r_i)}{(n - r_i)/(N - n - R + r_i)} \end{aligned}$$

This simplifies to $g(x) \approx \sum_i x_i \log \frac{N - n_i}{n_i}$ in the absence of relevance information, and approximates further to

Similarity and Ranking Operations. **Table 1.** Four popular methods for calculating the similarity between two objects. The binary vectors use a simple count of matching keywords, given their attribute vectors X and Y . The weighted vectors hold term weights calculated by an indexing algorithm; (w_X, w_Y) indicates the inner product of the vector weights, and $\sum w^2_A$ indicates the sum of squares of all weights of vector A .

Binary vectors		Weighted vectors	
Jaccard $\frac{ X \cap Y }{ X \cup Y }$	Dice $\frac{2 X \cap Y }{ X + Y }$	Jaccard $\frac{(w_X, w_Y)}{\sum w_X^2 + \sum w_Y^2 - \sum (w_X w_Y)}$	Dice $\frac{2(w_X, w_Y)}{\sum w_X^2 + \sum w_Y^2}$
cosine $\frac{ X \cap Y }{\sqrt{ X ^2 \times Y ^2}}$	overlap $\frac{ X \cap Y }{\min(X , Y)}$	cosine $\frac{(w_X, w_Y)}{\sqrt{\sum w_X^2 \sum w_Y^2}}$	overlap $\frac{(w_X, w_Y)}{\min(\sum w_X^2, \sum w_Y^2)}$

$$g(x) \approx \sum_i x_i \log \frac{N}{n_i} \text{ for } n_i \ll N, \text{ noting that the log}$$

factor in this last equation is identical to the common form for *inverse document frequency* (IDF). Once $g(X)$ is calculated, documents can be ranked by their relevance score.

The crucial factor in probabilistic retrieval is how to estimate *relevance*. The problem is that R is initially unknown, since there are no retrieved documents. To gain a notion of what may actually satisfy a user's information need, a common technique has been to estimate term relevance based on distributions of terms in the corpus. One approach is to simply use term IDF scores, and to assume that all query terms have an equal probability of appearing in relevant documents. Another approach is to use a mixture of two Poisson distributions to characterize the distribution of each term among relevant and irrelevant documents in the corpus.

Once the model has made some retrievals based on these relevance estimates, the user can provide *relevance feedback* by stating explicitly whether a retrieved document is relevant or irrelevant. This feedback is used to refine the model's term weight parameters incrementally. Relevance feedback is a direct approach to system personalization, but may not be practical with large corpora, or with systems with many (perhaps anonymous) users with different information needs. Relevance feedback also imposes a *burden of decision* that some users may wish to avoid, and its use reminds us that information retrieval is not a purely objective science: it is subject to often ill-defined information need. Users themselves may not be aware of their goal or able to describe their need, other than by a vague feeling of relevance.

Although probabilistic retrieval has been described here in its simplest form, further developments have addressed the troublesome assumption of *term independence* endemic to information retrieval, and accuracy of relevance scores has been improved by incorporating parameters for document length and *term frequency*.

Related Areas

Information retrieval deals primarily with textual information objects, although the advent of digital images and videos has led to techniques for quantifying the alikeness of non-textual media. Traditional textual similarity and ranking operations may be applied to non-textual media if they can first be interpreted into textual-symbolic form, such as by using machine-learning techniques to generate keyword descriptions of photographs. Whatever the source media, if symbolic (esp. alphanumeric) descriptive attributes can be extracted in a robust, consistent manner, then the media can be compared based on those attributes.

Alternatively, where many users share a common information space, they may add their own *tags* to information objects. Tagging is a form of *social filtering* where semantic relations between objects emerge from the collective valuations of a group of interested individuals. The resulting *tag cloud* acts as an organic index, preferentially retrieving objects strongly represented by a conjunction of query terms.

The World Wide Web, with its search engines, is by far the most popular information retrieval system: a text-based information medium with many millions of users and linked pages. Correspondingly, it has become an increasingly popular domain for information retrieval research and development. Although the term-based similarity methods discussed above

can be used to compare the content of documents, other effective methods have taken advantage of the semantic collaboration that gives the Web its structure: the many Web authors who link to pages that they consider relevant to their own. Web ranking algorithms (such as HITS and PageRank) observe which pages are highly-connected *hubs* and *authorities*; these pages are retrieved preferentially for a given query.

It is often useful to calculate the similarity between documents of a corpus, whether to find documents related to an exemplar document, or as a prelude to clustering operations. Since calculating similarity scores on demand in a large corpus can be expensive (particularly where corpus-wide sums are necessary, as with TF^*IDF), pair-wise similarity scores between documents may be stored in memory for later rapid retrieval. One approach is to create a similarity matrix—triangular if the similarity property is assumed to be commutative, but twice as large otherwise.

Another common approach is to store relations between documents in an *associative similarity network* (ASN). Documents act as the nodes of a network; links between pairs of documents represent relatedness, and are typically weighted with a real-valued similarity score. As many remotely-related documents could be linked with trivial weights, nodes are only linked for similarity scores above some threshold. In addition to fast nearest-neighbour retrieval, an ASN also provides opportunities for interactive navigation through the network, following a *semantic gradient descent* moving toward clusters of documents that better support a user's information need. ASNs can be seen as an example of the *cluster hypothesis*, which states that similar documents will be relevant to the same query: after a document has been retrieved with a term-based query, other documents similar to that document are already linked and immediately available for further inspection.

Key Applications

Classification; Thesaurus construction; Recommendation systems; Similarity (nearest-neighbour) search; Summarization and Information Filtering.

Cross-references

- ▶ [BM25](#)
- ▶ [Classification](#)
- ▶ [Clustering for Post-Hoc Information Retrieval](#)
- ▶ [Dimensionality Reduction](#)

- ▶ [Index Creation and File Structures](#)
- ▶ [Indexing and Similarity Search](#)
- ▶ [Multimedia Information Retrieval](#)
- ▶ [Probability Ranking Principle](#)
- ▶ [Relevance Feedback for Text Retrieval Model](#)
- ▶ [Stemming](#)
- ▶ [Term Weighting](#)
- ▶ [TF*IDF](#)
- ▶ [Web Information Retrieval Models](#)

Recommended Reading

1. Harman D. Ranking algorithms. In *Information Retrieval: Data Structures & Algorithms*, Chap. 14, W.B. and Frakes R. Baeza-Yates Prentice-Hall, Upper Saddle River, NJ, 1992, pp. 363–392.
2. Jones W.P. and Furnas G.W. Pictures of relevance: A geometric analysis of similarity measures. *J. Am. Soc. Inf. Sci.*, 38(6):420–442, 1987.
3. Rasmussen E.M. Clustering algorithms. In *Information Retrieval: Data Structures & Algorithms*, Chap. 16, W.B. Frakes and R. Baeza-Yates (eds.) Prentice-Hall, Upper Saddle River, NJ, 1992, pp. 419–442.
4. Salton G. and McGill M. *An Introduction to Modern Information Retrieval*. McGraw-Hill, New York, NY, 1983.
5. van Rijsbergen C.J. *Information Retrieval*. Butterworths, London, 1979.

Similarity in Video

- ▶ [Video Querying](#)

Similarity Measure

- ▶ [Image Retrieval](#)

Similarity-based Data Partitioning

- ▶ [Database Clustering Methods](#)

Simplicial Complex

ANDREW U. FRANK

Vienna University of Technology, Vienna, Austria

Synonyms

[CW complex](#); Polyhedron; Cell complex

Definition

A simplicial complex is a topological space constructed by gluing together dimensional simplices (points, line segments, triangles, tetrahedrons, etc.).

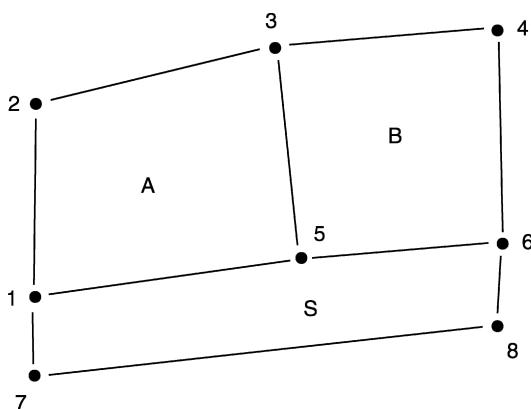
A simplicial complex K is a set of simplices k , which satisfies the two conditions:

1. Any face of a simplex in K is also in K
2. The intersection of any two simplices in K is a face of both simplices (or empty)

Historical Background

Raster (field) or vector (object) are the two dominant conceptualizations of space. Applications focusing on object with 2 or 3 dimensional geometry structure the storage of geometry as points, lines, surfaces, and volumes and the relations between them; a classical survey paper discussed the possible approaches mostly from the perspective of Computer Aided Design (CAD) where individual physical objects are constructed [10].

The representation of geographic information, e.g., maps, introduces consistency constraints between the objects; consider the sketch of a few cadastral parcels (lots) and the adjoining street (Fig. 1). Land, in this case 2 dimensional space, is divided into lots, such that the lots do not overlap and there are no gaps between them; this is called a partition (definition next section). Corbett [2] proposed to check that a sequence of line segments around a face closes and that the left neighbor of line segment and the right neighbors of the following line segment around a point is the same face; these two conditions are dual to each other (Fig. 2). This duality is the foundation of the DIME



Simplicial Complex. Figure 1. Cadastral parcels provide an example of a simplicial complex.

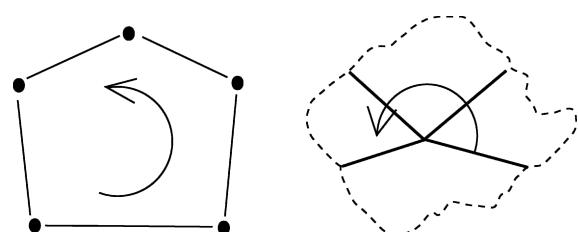
(dual independent map encoding) schema to store 2D line geometry for areas.

Every line of a graph, which represents a partition, is related to a start and an end point and to two adjacent faces (Fig. 3). Such data structures were typical for the 1980s; implemented originally with network and later relational DBMS. They did not perform acceptably fast with large Geographic Information System data, mostly because geometric operations do not translate to database operations directly (the so-called impedance mismatch of record oriented programming and tuple oriented database operations [7]), most obvious when checking geometric consistency. As late as 1985, all commercial programs to compute the overlay of two partitions, which is one of the most important operations in geographic information processing, failed.

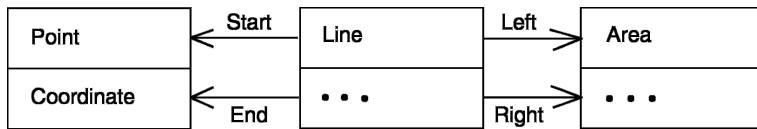
In 1986, Frank observed that simplicial (and possibly cell) complexes enforced exactly the consistency constraints required by the large class of applications that manage geometry as 2D or 3D partitions [5]. A commercial implementation became available, designed concurrently by Herring (then with Intergraph). Alternative approaches to manage the geometry of partitions without explicit representation of topology and to reconstruct topology when required were often used, but cause difficulties, because of the fundamental limitations of approximative numerical processing.

Foundations

Topology, specifically the theory of homotopy, provides the mathematical theory to program geometric operations. Homotopy captures the notion that multiple metric (coordinative) descriptions of a single geometry may be different but represent “essentially” the



Simplicial Complex. Figure 2. The two consistency checks: following the line segments around a face and following the line segments around a point.



Simplicial Complex. Figure 3. An UML object diagram for a database schema for partitions.

same geometry. [Figure 1](#) can be transformed continuously to [Fig. 4](#) but not to [Fig. 5](#).

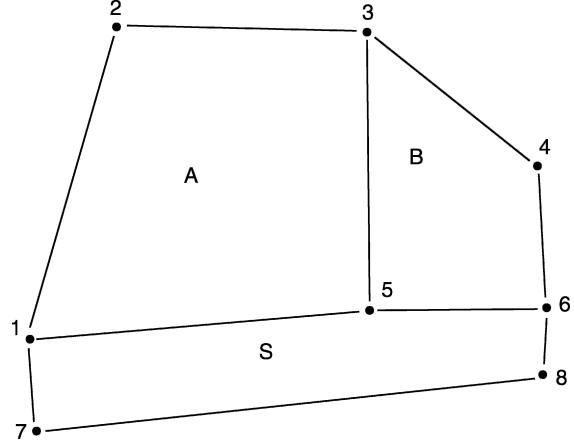
Homotopy creates equivalence classes for geometric figures. Many applications are interested in exactly these equivalence classes and benefit from the achieved abstraction that leaves out imprecisions caused, e.g., by measurements or approximative numerical processing.

Topology studies the invariants of space under continuous (homeomorphic) transformations, which preserve neighborhoods. Algebraic topology, also called combinatorial topology [1], studies invariants of spaces under homotopy with algebraic methods. The perspective of point set topology, which sees geometric figures as (infinite) sets of points is not practical for programming and the discretization of geometry achieved through algebraic topology is crucial: the unmanageable infinite sets are converted into countable objects, namely points, lines between points, and faces bounded by the boundary lines. Algebraic topology studies different “spaces” like [Fig. 4](#) and [Fig. 5](#) (both are embedded in ordinary 2D space, but the embedding is not in focus in algebraic topology).

The complexity of operations on arbitrary cells of a partition can be reduced by forcing a triangulation; all elements are then convex! [Figure 1](#) is a cell complex and the corresponding simplicial complex is [Fig. 6](#).

Algebraic topology studies simplices and their relations: A simplex is the simplest geometric figure in each dimension. A zero dimensional simplex (0-simplex) is a point, a one dimensional simplex (1-simplex) is a straight line segment, a two dimensional simplex (2-simplex) is a triangle, a three dimensional simplex (3-simplex) a tetrahedron, etc. $n + 1$ points in general position define an n -simplex. Each n -simplex consists of (is bounded) by $(n + 1)(n - 1)$ -simplices: a line (1-simplex) is bounded by 2 0-simplices (points), etc. Simplices can be oriented; the oriented 1-simplex 2–3 (in [Fig. 1](#)) is different from the oriented 1-simplex 3–2 ([Fig. 7](#)).

A k -simplicial complex K is a complex in which at least one simplex has dimension k and none a higher dimension. A homogeneous (or pure) k -complex K is a complex in which every simplex with dimension less

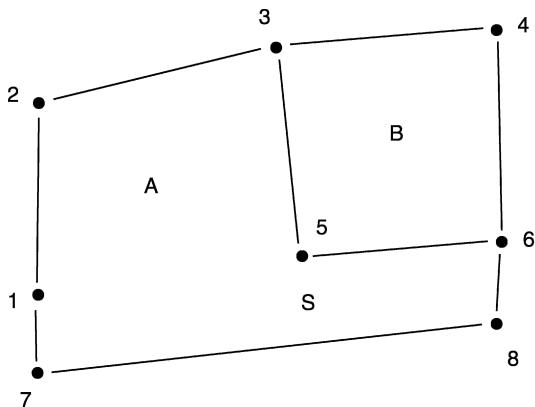


Simplicial Complex. Figure 4. A deformed, but homotopic, copy of [Fig. 1](#).

than k is the face of some higher dimension simplex in K . For example, a triangulation is a homogeneous 2-simplicial complex, a graph is a homogeneous 1-simplicial complex. Homogeneous simplicial complexes are models of partitions of space and used therefore to model geographic spatial data. Whitehead gave for so-called CW-complexes a slightly more general, more categorical definition mostly used in homotopy theory.

Four operations are important for simplicial complexes: the *closure* of a set of simplices S is the smallest complex containing all the simplices; it contains all the faces of every simplex in S . The *star* of a set of simplices S is the set of simplices in the complex that have simplices in S as faces. The *link* of a set of simplices S is a kind of boundary around S in the complex. The *skeleton* of simplicial complex K of dimension k is the subcomplex of faces of dimension $k-1$ in K .

Simplicial complexes can be represented as chains, which are lists of the ordered simplices included in the complex. Chains can be written as polynomials with integer factors for the simplices included in the complex, e.g., the 2-chain of the 2-complex in [Fig. 1](#) is $K = 1 \ A + 1 \ B + 1 \ S$.



Simplicial Complex. Figure 5. Metric is preserved, but the figure is not homotopic to Fig. 1, because elements are missing.

The boundary operator δ applied to a k -simplex gives the set of $k-1$ -simplices, which form the boundary of the simplex; for example, the boundary of a 1-simplex gives the two 0-simplices, which are start and end point of the line, one taken with positive, the other with negative orientation. The boundary operator is applied to a chain by applying it to every oriented simplex in the chain. The boundary of a closed simplicial complex is 0; in general, the boundary of the boundary is 0.

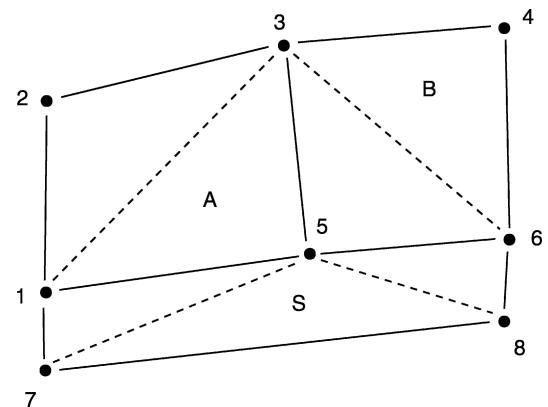
$$\begin{aligned}\delta A &= l_{12} + l_{23} + -l_{15} \\ \delta(\delta A) &= \delta l_{12} + \delta l_{23} + \delta l_{35} - \delta l_{15} \\ &= p_r - p_2 + p_2 - p_3 - p_s - p_r + p_s \\ &= 0\end{aligned}$$

The boundary operator is important to deduce the topological 4- and 9-intersection (Egenhofer) relations between two subcomplexes, of the same complex [3,4]. Chains and boundary operator are easy to implement with list operators and often it is sufficient to generalize the code for operations on polynomials.

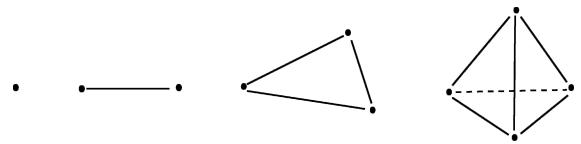
The theory of simplicial complexes can be generalized to cell complexes. Cells are homomorph to simplices, but can have arbitrary form; a 2-cell can have an arbitrary number of nodes in its boundary.

From an application point of view, it is often important that objects do not overlap and all of space is accounted for. The concept of a *partition* captures this idea; a partition of a space S is a set of subsets of the space, such that

- All subsets cover all of space (jointly exhaustive): $\bigcup_i s_i = S$



Simplicial Complex. Figure 6. The geometry of Fig. 1 triangulated.



Simplicial Complex. Figure 7. The simplices of 0, 1, 2, and 3 dimensions.

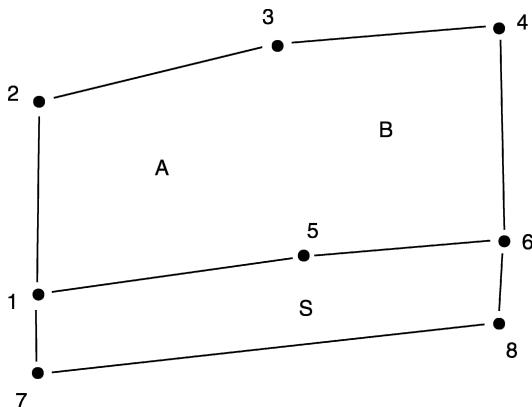
- No two subsets overlap (pairwise disjoint): $s_i \cap s_j = \emptyset$ for $i \neq j$.

These two properties are sometimes abbreviated as JEPD.

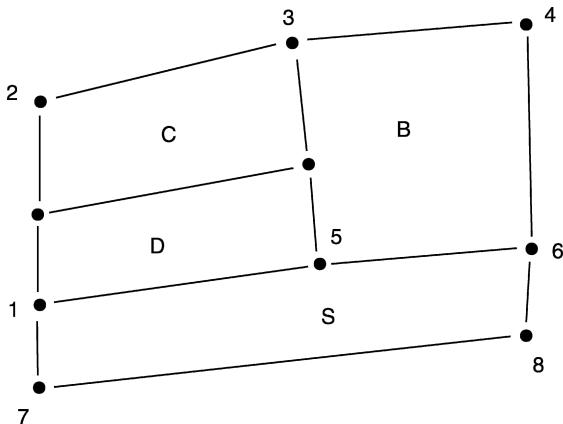
Partitions are changed by the Euler operations, *glue* and *split*, which maintain the Euler characteristic of the surface; the Euler characteristic is computed as $\chi = V - E + F$ where V is the number of nodes (vertices), E is the number of edges and F is the number of faces. From Fig. 1 with $\chi = 8 - 10 + 3 = 1$ merging two parcels obtains Fig. 8 with $\chi = 8 - 9 + 2 = 1$ or Fig. 9 where parcel A is split into parcel C and D with $\chi = 10 - 13 + 4 = 1$.

Consistency of these operations is difficult to check in cell complexes if “islands” occur as in Fig. 10, which is realistic for many application areas. The problem is avoided by triangulation and therefore simplicial complexes are an effective representation for maintainable geometric data describing partitions.

Simplicial complexes are triangulations of 2 dimensional space; they contain more objects than a partition represented as cells, but operations to maintain consistency in a triangulation are faster and simpler to program. The representation of a simplicial or cell complex



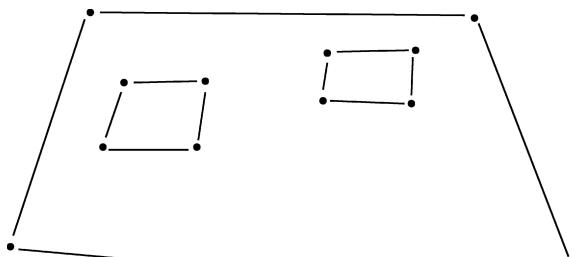
Simplicial Complex. Figure 8. A and B merged.



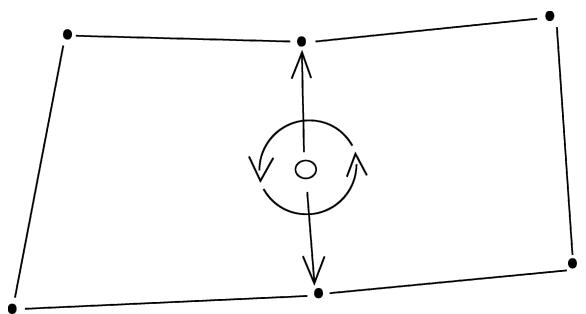
Simplicial Complex. Figure 9. A subdivided in C and D.

requires the explicit representation of the boundary and converse co-boundary relation. The schema used initially (Fig. 3) contains redundancy (which is used in Corbett's tests for consistency) and is therefore difficult to maintain. Popular today are schemes with half edges (Fig. 11), where a half-edge points to the starting node and the corresponding other half edge or quad edges [6] (Fig. 12), where each quad-edge points to the next quad-edge and either a boundary node or face; in a quad-edge structure, the boundary graph and its dual are maintained in a well-defined algebra with a single operation *splice*. For example, taking Fig. 1 as a boundary graph (primal) the dual is Fig. 13, which shows adjacency between faces.

Quad edges represent efficiently without redundancy a much larger universe, namely partitions of orientable manifold. The Euler operations *glue* and *split* can be efficiently implemented and maintain a simplicial or cell complex. The geometry can be represented as



Simplicial Complex. Figure 10. A parcel with "islands."



Simplicial Complex. Figure 11. Two half edges, pointing to adjacent nodes.

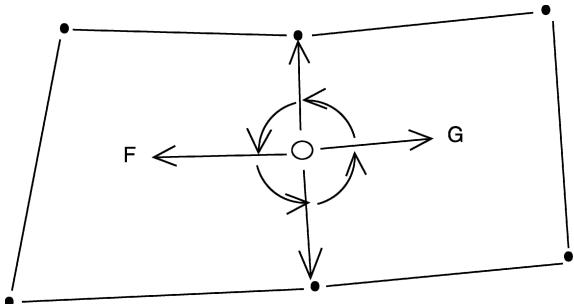
generalized maps, for which efficient implementation using relational databases has been reported [9].

Key Applications

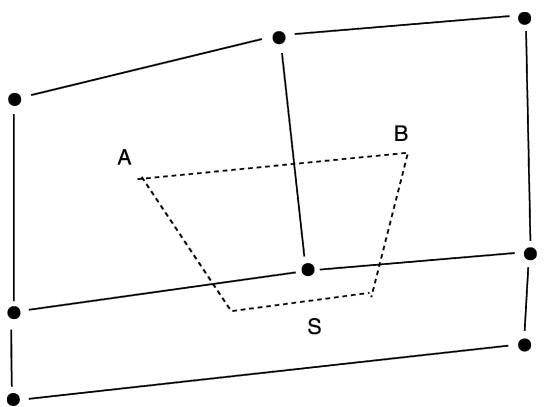
Many applications include geometric descriptions of objects; Computer Aided Design for mechanical and civil engineering are important, but also Geographic Information Systems, with many special applications like Utility Mapping for cities, Cadastral Maps to show ownership of land, but also car navigation systems, are popular examples.

Management of partitions is central for Geographic Information Systems (GIS); 2D partitions are wisely used for land ownership parcels, soil types, etc. Increasingly 3D models of cities and buildings are built to produce visualizations for virtual trips. Town planning applications expect that changes in 3D models over time can be visualized, which requires 4 (3 spatial plus one temporal) dimensions.

Management of the geometry of partitions of 3D space is important for CAD (Computer Aided Design), used for architecture, civil engineering but also mechanical engineering. Image processing intended to



Simplicial Complex. Figure 12. Four quad edges give one edge and point to adjacent nodes and faces.



Simplicial Complex. Figure 13. The dual graph of Fig. 1 (dashed) shows the neighbor relations.

produce 3D representations of the environment is using hierarchically structured partitions and needs effective operations to subdivide these.

A generalizable approach to storing and maintaining geometry in a database integrates for many application areas the treatment of geometric data with other data. Approaches based on the theory of simplicial or cell complexes are now available as plug-ins to convert general purpose DBMS to spatial databases. They replace earlier systems where geometric data was managed in proprietary file structures and the connection between geometry and descriptive data established only in the application program.

Future Directions

Besides efforts to enhance the performances of implementations three major research goals stand out:

1. Efficient solutions for 3D data; required for example to build 3D city models and to construct operations for consistently updating these [12]
2. Generalization to n -dimensions to include temporal data, especially 2 and 3 dimensional geometry and time required to include time related data, movement and, in general, processes in CAD and GIS applications [11]
3. Hierarchical structures to have partitions at one level of resolution (e.g., countries of the world) and then allow subdivision (e.g., regions, departments, counties, towns) [13]

A fully general application independent, n -dimensional and hierarchical representation that supports Euler operations effectively within data stored in a database is the implied goal of research in the first decade of the twenty-first century.

Cross-references

- [Geographic Information System](#)
- [Topological Data Models](#)
- [Topological Relationships](#)

Recommended Reading

1. Alexandrov P.S. Combinatorial Topology Volumes 1, 2 and 3. Dover Publications, Inc., Mineola, New York, 1960.
2. Corbett J.P. Topological Principles in Cartography, Bureau of the Census, US Department of Commerce, 1979.
3. Egenhofer M. and Herring J.R. A mathematical framework for the definition of topological relationships. In Proc. Fourth Int. Symp. on Spatial Data Handling, 1990.
4. Egenhofer M.J. and Franzosa R.D. On the equivalence of topological relations. Int. J. Geogr. Inf. Syst., 9(2):133–152, 1995.
5. Frank A.U. and Kuhn W. Cell graph: a provable correct method for the storage of geometry. In Proc. Second Int. Symp. on Spatial Data Handling, 1986.
6. Guibas L.J. and Stolfi J. A language for bitmap manipulation. ACM Trans. Grap., 1(3):191–214, 1982.
7. Härdler T. New approaches to object processing in engineering databases. In Proc. 1986 Int. Workshop on Object-Oriented Database Systems, 1986.
8. Levin B. Objecthood: an event structure perspective. In Proc. Chicago Linguistic Society 35, 1999, pp. 223–247.
9. Lienhardt P. Extensions of the notion of map and subdivision of a three-dimensional space. In Proc. 5th Annual Symp. on Theoretical Aspects of Computer Science, 1988.
10. Requicha A. Representation for rigid solids: theory, methods and Systems. ACM Comp. Surv., 12(4):437–464, 1980.
11. Sellis T, et al. (eds.) Spatiotemporal Databases: The Chorochronos Approach. LNCS, vol. 2520. Springer, Berlin, 2003.

12. Thompson R.J. Towards a Rigorous Logic for Spatial Data Representation. Doctoral thesis, Delft, NCG, 2007.
13. Timpf S. Hierarchical Structures in Map Series. Ph.D thesis, Technical University Vienna, Vienna, 1998.

$$\sum = \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_n \end{bmatrix} = diag(\sigma_1, \sigma_2, \dots, \sigma_m),$$

Simulated Data

- ▶ Synthetic Microdata

Single Instancing

- ▶ Deduplication

Single Instruction Multiple Data (SIMD) Parallelism

- ▶ Intra-operator Parallelism

Singular Value Decomposition

YANCHUN ZHANG, GUANDONG XU
Victoria University, Melbourne, VIC, Australia

Synonyms

SVD transformation; Latent semantic indexing; Principle component analysis

Definition

The SVD definition of a matrix is illustrated as follows [1]: For a real matrix $A = [a_{ij}]_{m \times n}$, without loss of generality, suppose $m \geq n$ and there exists SVD of A (shown in Fig. 1):

$$A = U \left(\begin{array}{cccc} \Sigma_1 & & & \\ & \ddots & & \\ & & \Sigma_r & \\ & & & 0 \end{array} \right) V^T = U_{m \times m} \sum_{m \times n} V_{n \times n}^T$$

where U and V are orthogonal matrices $U^T U = I_m, V^T V = I_n$. Matrices U and V can be respectively denoted as $U_{m \times m} = [u_1, u_2, \dots, u_m]_{m \times m}$ and $V_{n \times n} = [v_1, v_2, \dots, v_n]_{n \times n}$, where $u_i, (i = 1, \dots, m)$ is a m -dimensional vector $u_i = (u_{1i}, u_{2i}, \dots, u_{mi})^T$ and $v_j, (j = 1, \dots, n)$ is a n -dimensional vector $v_j = (v_{1j}, v_{2j}, \dots, v_{nj})^T$. Suppose $\text{rank}(A) = r$ and the single values of A are diagonal elements of Σ as follows:

where $\sigma_i \geq \sigma_{i+1} > 0$, for $1 \leq i \leq r-1$; $\sigma_j = 0$, for $j \geq r+1$, that is

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r \geq \sigma_{r+1} = \dots = \sigma_n = 0$$

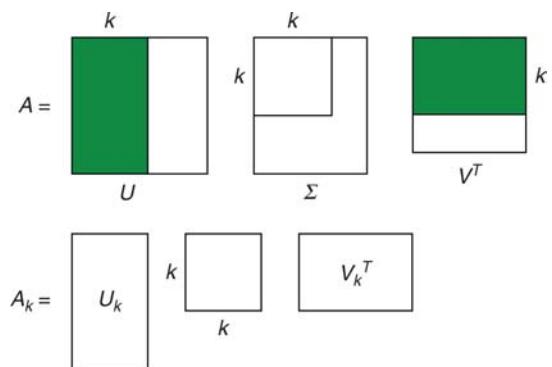
For a given threshold ε ($0 < \varepsilon < 1$), choose a parameter k such that $(\sigma_k - \sigma_{k+1})/\sigma_k \geq \varepsilon$. Then, denote $U_k = [u_1, u_2, \dots, u_k]_{m \times k}$, $V_k = [v_1, v_2, \dots, v_k]_{n \times k}$, $\sum_k = diag(\sigma_1, \sigma_2, \dots, \sigma_k)$, and

$$A_k = U_k \sum_k V_k^T$$

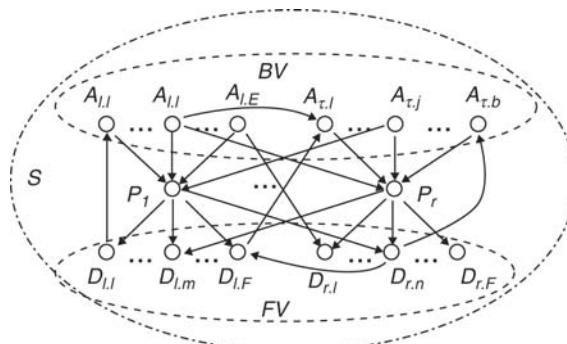
As known from the theorem in algebra [1], A_k is the best approximation matrix to A and conveys main and latent information among the processed data. This property makes it possible to find out the underlying semantic association from original feature space with a dimensionality-reduced distance computational cost, in turn, is able to be used for latent semantic analysis.

Key Points

Singular Value Decomposition (SVD) algorithm could be considered as a useful means for applications of data engineering and knowledge discovery, such as Web search, image and document retrieval and Web data mining. For example, finding the closely relevant pages to a given page can be carried out by manipulating



Singular Value Decomposition. Figure 1. SVD transformation of matrix and its approximation.



Singular Value Decomposition. Figure 2. Page source structure for the given page u .

SVD operation on a constructed page source to reveal the latent linkage relationships from them [3]. In order to avoid high cost of similarity computations and keep the minimum loss of linkage information contained in the feature space, SVD algorithm is applied to not only reduce the dimensionality of original feature, which leads to less computational costs, but also capture the semantic similarity among web pages, which is unseen intuitively. The algorithm is working as follows: (i) construct a web page space A (page source) for the given page u from link topology on the web. The page source is represented as a directed graph with edges indicating hyperlinks and nodes standing for web objects (shown in Fig. 2); (ii) since the topological relationships amongst the page source is expressed in a linkage matrix, manipulating SVD results in decomposition of original feature space $A = U_{m \times m} \Sigma_{m \times n} V_{n \times n}^T$; (iii) From the SVD theorem, the best approximation matrix A_k contains main linkage information among the pages, and makes it possible to filter those irrelevant pages; (iv) by selecting a threshold of similarity, the relevant pages in page source to the given page are eventually found.

While SVD algorithm is usually used in conventional latent semantic analysis (LSA) techniques, some variants of LSA have been proposed recently in the context of Web information processing and text mining. Apart from the difference at theoretical formulation, the common characteristics of these methods are to map the original feature space, which is utilized to model the co-occurrence observation, into a new dimensionality-reduced feature space, and maintain the maximum approximation of the original feature distance with the converted feature space. For example, Probabilistic

Latent Semantic Analysis (PLSA) model is an representative of such kinds of approaches [2]. For instance, [3] proposed a PLSA-based Web usage mining approaches for Web recommendation. In this collaborative recommendation scheme, user task-oriented access patterns are extracted from Web log files, in turn, are used to predict user's likely interested Web content via referring the navigational preference of other users, who exhibit like-minded access task.

Cross-references

- ▶ Database Clustering Methods
- ▶ Principal Component Analysis

Recommended Reading

1. Datta B. Numerical Linear Algebra and Application. Brooks/Cole Publishing Company, Pacific Grove, CA, 1995.
2. Hofmann T. Latent semantic models for collaborative filtering. ACM Trans. Inf. Syst., 22(1):89–115, 2004.
3. Zhang Y., Yu J.X., and Hou J. Web Communities: Analysis and Construction. Springer, Berlin, 2006.

Sketch

- ▶ AMS Sketch
- ▶ Structure Indexing

SMI-S

- ▶ Storage Management Initiative-Specification

Snapshot

- ▶ Point-in-Time Copy (PiT Copy)

Snapshot Data

- ▶ Atelic Data

Snapshot Equivalence

CHRISTIAN S. JENSEN¹, RICHARD T. SNODGRASS²

¹Aalborg University, Aalborg, Denmark

²University of Arizona, Tucson, AZ, USA

Synonyms

Temporally weak; Weak equivalence

Definition

Informally, two tuples are *snapshot equivalent* or *weakly equivalent* if all pairs of timeslices with the same time instant parameter of the tuples are identical.

Let temporal relation schema R have n time dimensions, D_i , $i = 1, \dots, n$, and let τ^i , $i = 1, \dots, n$ be corresponding timeslice operators, e.g., the valid timeslice and transaction timeslice operators. Then, formally, tuples x and y are snapshot equivalent if

$$\begin{aligned} \forall t_1 \in D_1 \dots \forall t_n \in D_n (\tau_{t_n}^n(\dots(\tau_{t_1}^1(x))\dots) \\ = \tau_{t_n}^n(\dots(\tau_{t_1}^1(y))\dots)) \end{aligned}$$

Similarly, two relations are snapshot equivalent or weakly equivalent if at every instant their snapshots are equal. Snapshot equivalence, or weak equivalence, is a binary relation that can be applied to tuples and to relations.

Key Points

The notion of weak equivalence captures the information content of a temporal relation in a point-based sense, where the actual timestamps used are not important as long as the same timeslices result. For example, consider the two relations with just a single attribute: $\{(a, [3,9])\}$ and $\{(a, [3,5]), (a, [6,9])\}$. These relations are different, but snapshot equivalent.

Both “snapshot equivalent” and “weakly equivalent” are being used in the temporal database community. “Weak equivalence” was originally introduced by Aho et al. in 1979 to relate two algebraic expressions [1,2]. This concept has subsequently been covered in several textbooks. One must rely on the context to disambiguate this usage from the usage specific to temporal databases. The synonym “temporally weak” does not seem intuitive—in what sense are tuples or relations weak?

Cross-references

- ▶ Temporal Database
- ▶ Time Instant
- ▶ Timeslice Operator
- ▶ Transaction Time
- ▶ Valid Time
- ▶ Weak Equivalence
- ▶ Point-Stamped Temporal Models

Recommended Reading

1. Aho A.V., Sagiv Y., and Ullman J.D. Efficient optimization of a class of relational expressions. *ACM Trans. Database Syst.*, 4(4):435–454, 1998.
2. Aho A.V., Sagiv Y., and Ullman J.D. Equivalences among relational expressions. *SIAM J. Comput.*, 8(2):218–246, 1979.
3. Gadia S.K. Weak temporal relations. In Proc. 4th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1985, pp. 70–77.
4. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In *Temporal Databases: Research and Practice*, O. Etzion, S. Jajodia, S. Sripana (eds.). LNCS 1399, Springer-Verlag, 1998, pp. 367–405.

Snapshot Isolation

ALAN FEKETE

University of Sydney, Sydney, NSW, Australia

Synonyms

SI; Row-versioning

Definition

Snapshot Isolation is a multi-version concurrency control approach that is widely used in practice. A transaction T that operates under Snapshot Isolation never observes any effects from other transactions that overlap T in duration; instead T sees values as if it were operating on a private copy or snapshot of the database, reflecting all other transactions that had committed before T started. In Snapshot Isolation, the system will not allow both of two transactions to commit if they overlap in duration and modify the same data item. Snapshot Isolation prevents many well-known anomalies (such as Lost Updates and Inconsistent Reads) that are also prevented by Serializability, but it does not guarantee that all executions will be Serializable. Snapshot Isolation allows reads to occur without delay or blocking caused by

concurrent updates, and also updates are never blocked by concurrent readers, so Snapshot Isolation often gives the transactions better throughput than traditional concurrency control based on Two-Phase Locking.

Historical Background

The idea of providing a multi-version concurrency control algorithm based on reading from a private snapshot has been wide-spread since at least 1982, Chan et al. [4] provided a detailed discussion of how this could work, considering many practical issues. However, in this early period, the idea was only considered for read-only transactions. That is, any transaction that included writes was treated with a different concurrency control, in order to ensure serializability.

In 1995, Berenson et al. [2] introduced the term Snapshot Isolation, and explained the algorithm for transactions that include write operations. They did so in the context of a paper about ambiguities in how the SQL Standard defines Isolation. As well as describing the algorithm, this paper showed that it could allow non-serializable executions, though it did not allow any of the particular erroneous phenomena that had been considered in the SQL Standard. Also in 1995, Oracle 7 introduced the use of the algorithm when transactions request “Isolation Level Serializable” [9]; previous releases of Oracle had not provided any transaction-level consistency, but rather they had each SQL statement within a transaction see a different state of the database. In 1999, Oracle obtained a US patent (number 5870758) on their approach.

Considerable research has been devoted to understanding the properties of executions allowed by Snapshot Isolation. Bernstein et al. [3] showed how to reason about whether transactions preserve individual integrity constraints when run with Snapshot Isolation. In 2005, Fekete et al. [8] published a theory to show which sets of transactions are guaranteed to run serializably on a DBMS platform that uses Snapshot Isolation (and therefore that every possible integrity constraint is preserved); some of these checks were automated by Jorwekar et al. [10]. Several papers [3,7] look at issues that arise when some transactions run with Snapshot Isolation, while other transactions use other concurrency control techniques. An abstract characterization of the isolation provided by Snapshot Isolation was suggested by Adya [1].

Since 2000, the issues of storing replicated data in sites each using Snapshot Isolation has been studied extensively. Different approaches are explored by many researchers [5,6,11,12,13,16].

Foundations

Like other concurrency control techniques, Snapshot Isolation (hereafter abbreviated SI) responds to requests from clients for reading and writing data items that are stored by the DBMS engine. Each request could be either performed immediately or delayed for a while, or the requesting transaction might be aborted. SI is a multi-version mechanism, so when performing a request to read an item, the engine might return a value other than the current value of the item (that is, it might return a value that had been written in some earlier transaction, not necessarily the value written by the transaction that most recently altered the item).

The SI mechanism is defined by two properties. The first property (that explains the term “Snapshot”) determines which value is to be returned in a read operation. If transaction T reads a data item x , then the value returned by the system is whichever value was written by the most recently committed transaction, among all the transactions that wrote the item x and also committed before T started. There is one exception to this rule: if T itself writes the item x , and later requests to read x , then T will see the value it wrote itself. The second property of SI is sometimes called “First Committer Wins”; it requires the engine to prevent the situation where there are two concurrent (Transactions T and U are concurrent if their duration overlaps, that is there is some intersection between the interval from the start of T to its completion, and the interval from the start of U to its completion.) transactions that both write to the same item, and that both commit.

To illustrate the way SI controls concurrency, consider the sequence of operations shown in Schedule 1. In presenting this and later schedules, this entry uses the standard notation where each operation has a subscript that indicates which transaction performs the operation, so r_i is a read within T_i . As well as subscripts on the operations, each data item has versions that are indicated by subscripts, where the version of item x produced by transaction T_i is represented as x_i . Thus when T_5 writes x , this is indicated by $w_5[x_5]$, and if T_3 later does a read of x that

returns the value that was written by T_5 , one includes $r_3[x_5]$ in the schedule. (Readers might mistakenly think that the subscript indicates the version order; thus they think that x_2 represents the second version of x . This is not the case in the notation used here.) The usual notation is extended by representing the start of transaction T_i as b_i (while the completion of the transaction is c_i in the case of a commit, or a_i when the transaction aborts)

$$\begin{aligned} b_1 r_1[x_0] r_1[y_0] c_1 b_2 w_2[x_2] a_2 b_3 r_3[x_1] r_3[y_1] w_3[x_3] \\ b_4 r_4[x_1] r_4[y_1] w_3[y_3] r_3[x_3] c_3 c_4 \end{aligned} \quad (1)$$

In Schedule 1, notice that the snapshot used by T_4 includes the changes made by T_1 (which committed before T_3 started), but not those made by T_2 which aborted, nor those made by T_3 which is concurrent with T_4 . In particular, when T_4 reads x , it sees the version x_1 which was written by T_1 , even though there is already a more recent version x_3 . In this schedule one also can find an example where T_3 reads x twice, having modified the item in between; the second time T_3 reads x , it sees its own version.

SI is an attractive concurrency control mechanism for many reasons. It usually performs well, and in particular it does not suffer from the delays that can reduce throughput in locking-based concurrency control. For example, if a large slow transaction T is reading many items, in order to calculate some complicated statistics, traditional two-phase locking takes read locks on many items, and holds these while T is running; through this long period, other transactions which want to change those items will be blocked if locking is used for concurrency control. Under SI, in contrast, the large slow transaction T does not take read locks, and concurrent transactions can update the items.

The database literature has identified a number of anomalies that can occur from uncontrolled concurrency. For example, the Inconsistent Read phenomenon happens when a transaction T sees some but not all of the changes made by another transaction U . Under SI, this can't happen: if U committed before T starts, then the effects of U are all in the snapshot used when T reads, while if U is still running when T starts, then none of the changes made by U are in the snapshot. For example, in Schedule 1 the snapshot used when T_4 reads includes both changes made by T_1 (to x and also to y) and none of the changes made by T_3 . Another famous phenomenon is Lost Update. An

example of this occurs when two transactions both read an item, and both produce new values that increment what they read; if both these transactions commit, the final value of the item will be incremented by one instead of by two (as would happen in a serial, non-interleaved execution). If the database platform is using SI, the First Committer Wins property will prevent Lost Update (as one of the transactions will be required to abort). For example, in Schedule 2, T_2 is not allowed to commit (since T_1 and T_2 are concurrent and both have written the same item x , the property says that they cannot both commit).

$$b_1 r_1[x_0] w_1[x_1] b_2 c_1 r_2[x_0] w_2[x_2] a_2 \quad (2)$$

Despite preventing the well-known concurrency control anomalies, SI does not ensure that all executions are serializable. Schedule 3 shows an anomaly called “Write Skew.”

$$b_1 r_1[x_0] r_1[y_0] b_2 r_2[x_0] r_2[y_0] w_1[x_1] w_2[y_2] c_1 c_2 \quad (3)$$

In Schedule 3, the First Committer Wins property is not effective, because the concurrent transactions do not have any item that both of them write (T_1 writes x and T_2 writes y). The lack of serializability in this schedule can result in data corruption. For example, suppose x and y are data items that represent the balance in two different bank accounts, and suppose a business rule requires the sum of the balances to be positive. Suppose initial values are $x_0 = 100$ and $y_0 = 200$, and T_1 is reducing x by 150 (aborting if it sees insufficient funds in the combined balance), while T_2 reduces y by 175 (again, aborting if there is not enough in the total of the balances). One sees that each transaction, run alone, preserves the business rule; however the Schedule 3 is possible with SI and yet it produces a final state where x is -50 , and y is 25 , violating the integrity of the data according to the business rule.

Because many developers think that correct isolation ought to be what SI does (namely, a transaction does not see any effects of concurrent transactions), it is worth explaining why this is not so. Correct isolation (“serializable” execution) means that the outcome is just like in a serial or batch execution. In a batch execution, between any pair of transactions, one will come first, and so the other will see its effects. Thus if neither of two transactions sees the other, this is not like a batch execution. The Write Skew example shows that two transactions may each decide to take some

action like removing money from a bank account, where it is acceptable for one to make the change, but not when the other has already done so. When neither sees the other, they might both make the change and commit.

The Schedule 3 is not serializable, and this can be proved because it has a multi-version serialization graph with a cycle. In the literature, there are several variant definitions of multi-version serialization graph (MVSG) for a schedule; this entry uses the one in the text by Weikum and Vossen [15]. MVSG is defined for a given schedule and a version order \ll which relates every pair of versions of the same data item. When working with SI, the version order is always taken as the order of the commits of the transactions that wrote the versions; that is one defines $x_i \ll x_j$ when T_i commits before T_j . MVSG has nodes for the transactions, and there is an edge from T_i to T_j in the following three circumstances: (i) the schedule contains $r_j[x_i]$ for some item x , (ii) $x_i \ll x_j$ and the schedule contains $r_k[x_j]$ for some x and k , or (iii) the schedule contains $r_i[x_k]$ and $x_k \ll x_j$ for some x and k . It turns out that for understanding the behavior of SI, it is important to pay attention to particular edges in the MVSG: those which go between concurrent transactions. Call such an edge vulnerable, and draw it with a dashed line in the multi-version serialization graph. Notice that the First Committer Wins rule means that there can never be a vulnerable edge between two transactions if there is some data item to which both transactions write. The Snapshot property means that if there is an edge from T_i to T_j because of an operation $r_j[x_i]$, then T_i must have committed before T_j started and so the edge is not vulnerable. Thus the only vulnerable edges arise from conflicts where one transaction reads an item which the concurrent transaction writes. The MVSG for the Schedule 3 above is in Fig. 1.

Even though SI can allow executions that are not serializable, these executions are not observed often. There are some sets of application programs which never give rise to a non-serializable execution when running with SI as the concurrency control mechanism. For example many of the standard benchmark



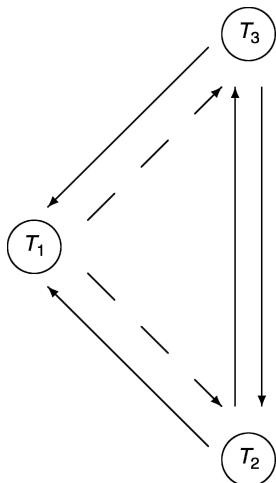
Snapshot Isolation. Figure 1. MVSG for Schedule 3.

suites, such as TPC-C [14], generate only serializable schedules. It can be proved [8] that in any schedule allowed by SI, if there is a cycle in the MVSG, then the cycle contains two *consecutive* vulnerable edges. Given a set of transactions T_1, T_2 etc, one can draw a static dependency graph SDG, which is a directed graph whose nodes are transactions, with an edge from T_i to T_j if it is possible to find a schedule h with some of these transactions, so that $MVSG(h)$ has an edge from T_i to T_j . Furthermore, one says that the edge in SDG is *vulnerable* if there is a schedule h where the edge in $MVSG(h)$ is vulnerable. Note that $MVSG(h)$ depends on the schedule h which shows how the transactions interleave, but SDG can be found from the set of separate transactions. Because for any schedule h , $MVSG(h)$ is a subset of SDG, it follows that if SDG has no cycle with consecutive vulnerable edges, then $MVSG(h)$ also has no cycle with consecutive vulnerable edges, and so h is serializable. Thus, a set of transactions will always interleave in serializable executions under SI, provided that SDG does not have any cycle with consecutive vulnerable edges.

As an example, consider the transactions in Fig. 2. For these transactions, the SDG is shown in Fig. 3. The only vulnerable edges in this SDG are from T_1 to T_2 and from T_1 to T_3 . The edge from T_2 to T_1 is not vulnerable because T_1 has no write operations (and under SI, a vulnerable edge can only come from a read-to-write conflict), and similarly T_3 to T_1 is not vulnerable. The edges between T_2 and T_3 are not vulnerable (in either direction) because both transactions write the item x , and so the First Committer Wins property of SI prevents these transactions both committing if they are concurrent. Thus there are no consecutive vulnerable edges at all in SDG, and so every execution of T_1, T_2 and T_3 will be serializable when they run on a platform using SI for concurrency control. To use these ideas in practice, one needs to deal with application code that contains parameterized SQL statements, and complicated control flow; the techniques needed are discussed in [8, 10].

$$\begin{aligned} T_1 &= r[x]r[y]r[z] \\ T_2 &= r[x]r[y]w[x]w[y] \\ T_3 &= r[x]r[z]w[x]w[z] \end{aligned}$$

Snapshot Isolation. Figure 2. Transactions with every execution serializable.



Snapshot Isolation. Figure 3. SDG for Transactions from Fig. 2.

What can the database administrator do if they have an application which will run on a platform where SI is the concurrency control mechanism, and yet the application is made up of transactions that are not guaranteed to have serializable executions on such a platform? The natural approach is to alter the application code, without changing the meaning of each transaction, so that the changed transactions are certain to execute serializably. This means changing programs so as to make some edges from the SDG be not vulnerable. Two techniques are known to change transactions T_i and T_j where the edge from T_i to T_j is vulnerable. One can materialize the conflict, by creating a new table called say Conflict, and including in both T_i and T_j an update of a particular row in this table. Alternatively, one can sometimes leave T_j unaltered, and introduce an identity write into T_i . That is, perform “SET $x=x$ ” in an UPDATE statement which is added to the code of T_i , to affect whichever data item (row) is the one which T_i reads and T_j writes.

In conclusion, SI is a concurrency control mechanism that has many attractive features. It usually gives quite good throughput, since a read operation is never delayed by other transactions that are changing the data, and updates are not delayed when other transactions have read the data they want to change. The outdated versions that are used in SI, to respond to read requests, are often available anyway, because they are kept to support rollback recovery. SI prevents many bad executions; it can't suffer from Lost Update or Inconsistent Read or Phantoms. The way SI works is

easy to understand, and indeed many articles have just assumed that being “isolated” means “not seeing any changes made by concurrent transactions” (as happens in SI). However, SI does not enforce that every execution will be serializable. Developers and users need to be aware that when SI is used, it is possible that transactions can interleave in ways that make the data invalid according to some business rule which is obeyed by every transaction running alone.

Key Applications

Snapshot Isolation is used as a concurrency control mechanism in a wide range of common platforms. For example, Microsoft SQL Server 2005 offers it when a user chooses to invoke “SET TRANSACTION ISOLATION LEVEL SNAPSHOT.” It is similarly available as a separate isolation level in Interbase and Oracle Berkeley DB. Other platforms, such as PostgreSQL (since version 7) and Oracle, use SI when the client chooses “SET ISOLATION LEVEL SERIALIZABLE” even though SI does allow non-serializable executions. SI is very useful in managing replicated data. One can combine individual databases which use SI, to act transparently as a global one-copy database. This is easier than to combine traditional locking databases to provide one-copy serializability. Many research prototypes combine SI with consistent replication, however these ideas are not widely used in practice yet.

Future Directions

The main focus of current research with SI is in replicated data management. There are many issues that arise when data are replicated between sites some or all of which use SI rather than traditional locking for local concurrency control. Many different systems have been designed and evaluated but no clear winner has yet emerged, so research is continuing. Another topic that needs more understanding is the performance of SI, in particular to understand which characteristics of an application domain make SI perform better or worse than other concurrency control techniques.

Cross-references

- ▶ [Concurrency Control – Traditional Approaches](#)
- ▶ [Consistency Models for Replicated Data](#)
- ▶ [Multi-Version Serializability and Concurrency Control](#)
- ▶ [Replication for Scalability](#)

- ▶ [Serializability](#)
- ▶ [SQL Isolation Levels](#)

Recommended Reading

1. Adya A. Weak consistency: a generalized theory and optimistic implementations for distributed transactions (PhD thesis). Technical Report MIT/LCS/TR-786, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, USA, 1999.
2. Berenson H., Bernstein P.A., Gray J., Melton J., O'Neil E.J., and O'Neil P.E. A critique of ANSI SQL isolation levels. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 1–10.
3. Bernstein A.J., Lewis P.M., and Lu S. Semantic conditions for correctness at different isolation levels. In Proc. 16th Int. Conf. on Data Engineering, 2000, pp. 57–66.
4. Chan A., Fox S., Lin W.-T.K., Nori A., and Ries D.R. The implementation of an integrated concurrency control and recovery scheme. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1982, pp. 184–191.
5. Daudjee K. and Salem K. Lazy database replication with snapshot isolation. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 715–726.
6. Elnikety S., Zwaenepoel W., and Pedone F. Database replication using generalized snapshot isolation. In Proc. 24th IEEE Symp. on Reliable Dist. Syst., 2005, pp. 73–84.
7. Fekete A. Allocating isolation levels to transactions. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 206–215.
8. Fekete A., Liarokapis D., O'Neil E., O'Neil P., and Shasha D. Making snapshot isolation serializable. ACM Trans. Database Syst., 30(2):492–528, 2005.
9. Jacobs K. Concurrency control: transaction isolation and serializability in SQL92 and Oracle7. Technical Report A33745 (White Paper), Oracle Corporation, 1995.
10. Jorwekar S., Fekete A., Ramamritham K., and Sudarshan S. Automating the detection of snapshot isolation anomalies. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1263–1274.
11. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based data replication providing snapshot isolation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 419–430.
12. Plattner C. and Alonso G. Ganymed: scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004, pp. 155–174.
13. Schenkel R. and Weikum G. Integrating snapshot isolation into transactional federation. In Proc. Int. Conf. on Cooperative Inf. Syst., 2000, pp. 90–101.
14. Transaction Processing Performance Council. TPC Benchmark C Standard Specification, Revision 5.0. 2001. URL: <http://www.tpc.org/tpcc/>
15. Weikum G. and Vossen G. Transactional information systems: Theory, algorithms, and the practice of concurrency control and recovery. Morgan Kaufmann, Los Altos, CA, USA, 2002.
16. Wu S. and Kemme B. Postgres-R(SI): combining replica control with concurrency control based on snapshot isolation. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 422–433.

SNIA

- ▶ [Storage Network Industry Association](#)

Snippet

MARCUS HERZOG^{1,2}

¹Vienna University of Technology, Vienna, Austria

²Lixto Software GmbH, Vienna, Austria

Synonyms

[Web widget](#); [Macro](#); [Module](#); [Capsule](#); [Mini](#); [Flake](#)

Definition

A snippet is a chunk of re-usable source code. In the context of Web programming, a snippet refers to a chunk of re-usable HTML source code, along with all relevant resources such as stylesheets and scripts applied within the context of the snippet. In the context of Web information extraction, a snippet is a subset of the available information items that can be extracted from the Web page.

Key Points

The term snippet originates from the domain of text editors, where snippets refer to chunks of source code which can be organized for copy and paste usage. Snippet management allows for viewing, editing, sorting, and storing snippets in a repository of re-usable source code fragments. The overall goal of snippets is to ease the process of writing code by reducing the manual effort to type in source code and to re-use existing lines of code.

Snippets can be classified according to the complexity of the interaction process: static, dynamic, and scriptable snippets. A static snippet is a fixed chunk of text that can be inserted at the cursor position. This operation is similar to a cut-and-paste operation well known from text editors. Dynamic snippets contain some dynamic elements which are filled in on insertion of the snippet into the main document. Scriptable snippets take this dynamic concept one step further by not only allowing for filling in placeholders, but by providing means to compute the values of placeholders, e.g., by applying a transformation operation on a placeholder value.

In Web programming, snippets are often used when assembling a web page from pre-existing building blocks. This is very popular in constructing social network home pages or other types of personal Web 2.0 applications such as Blogs. In this context snippets are often referred to as e.g., Web widgets, minis, or flakes, depending on the framework in which the snippet is programmed. In Web programming a snippet is already more like a mini application which can be re-used in the context of a Web application, e.g., a portal such as iGoogle or MyYahoo.

In Web data extraction [1] the concept of snippet is used to refer to a particular part of the Web page which is extracted and transformed into an information item. Here the emphasis is on the re-use of existing data or content which is transformed into a presentation-independent representation, e.g., XML document format. The goal is to re-use exiting data in the context of new applications which assemble data snippets from various sources and provide additional value by relating the content extracted from these independent sources.

Cross-references

- ▶ [Blogging](#)
- ▶ [Re-Usable Code](#)
- ▶ [Re-Usable Information Item](#)
- ▶ [Web Programming](#)

Recommended Reading

1. Baumgartner R., Flesca S., and Gottlob G. Visual web information extraction with lixto. In Proc. 27th int. Conf. on Very Large Data Bases, 2001, pp. 119–128.

Definition

A *snowflake schema* has one “central” table whose primary key is compound, i.e., consisting of multiple attributes. Each one of these attributes is a foreign key to one of the remaining tables, which may, in turn, have some of its non-key attributes each be a foreign key to yet another, different table. This continues recursively with the remaining tables, until they are exhausted, forming chains or trees of foreign key dependencies rooted at the “central” table, i.e., each table in the schema (except the “central” table) is pointed to by exactly one such foreign key. (In the above, without loss of generality, we make the assumption that all tables except the “central” table have simple primary keys. This is usually the case in almost all practical situations, as for efficiency, these keys are often generated, *surrogate keys*.)

Key Points

Many data warehouses (see definitional entry for *Data Warehouse*) that represent the multidimensional conceptual data model in a relational fashion [1,2] store their primary data as well as the data cubes derived from it in snowflake schemas, as an alternative to *star schemas*. As in star schemas, the “central” table and the remaining tables of the definition above correspond, respectively, to the *fact table* and the *dimension tables* that are typically found in data warehouses. Each *fact* (tuple) in the fact table consists of a set of numeric *measures*, comprising the objects of analysis, and a set of *dimensions*, which uniquely determine the set of measures. The remaining tables store the attributes of the aforementioned dimensions at different levels of granularity.

Unlike star schemas, snowflake schemas can explicitly capture hierarchies in the dimensions, with each table in each chain (or tree path) of foreign key dependencies corresponding to one level of one such hierarchy. For instance, dimension *Store* in the example below contains values at different levels of detail, forming the hierarchy *Street*→*City*→*State*. On the contrary, star schemas capture all levels of a hierarchical dimension in a single, de-normalized table. Starting from a star schema (usually in Second Normal Form), one may generate the corresponding snowflake schema (usually in Third Normal Form at least) by normalization, decomposing the dimensions into multiple tables. Accordingly, star schemas lend themselves to simpler and usually faster

Snowflake Join Schema

- ▶ [Snowflake Schema](#)

Snowflake Schema

KONSTANTINOS MORFONIOS, YANNIS IOANNIDIS
University of Athens, Athens, Greece

Synonyms

- ▶ [Snowflake join schema](#)

queries, while snowflake schemas are easier to maintain and require less space.

For example, consider a data warehouse of a retail chain with many stores around a country. The dimensions may be the products sold, the stores themselves with their locations, and the dates, while the numeric measures may be the number of items and the total monetary amount corresponding to a particular product sold in a particular store on a particular date. The relevant snowflake schema, with the product, store, and date dimensions normalized, is shown below, where *SalesSummary* is the fact table, primary keys are in italics, and each attribute of the fact-table primary key as well as each non-key ‘Id’ attribute of the other tables is a foreign key.

```
SalesSummary (ProductId, StoreId, DateId,  
 NumOfItems, TotalAmount)  
Product (ProductId, ProdName, Prod-  
Descr, CategoryId, UnitPrice)  
Category (CategoryId, CategoryDescr)  
Store (StoreId, StreetId)  
Street (StreetId, Street, CityId)  
City (CityId, City, StateId)  
State (StateId, State)  
Date (DateId, Date, MonthId)  
Month (MonthId, Month, YearId)  
Year (YearId, Year)
```

Cross-references

- ▶ [Cube Implementations](#)
- ▶ [Data Warehouse](#)
- ▶ [Dimension](#)
- ▶ [Hierarchy](#)
- ▶ [Measure](#)
- ▶ [Multidimensional Modeling](#)
- ▶ [Star Schema](#)

Recommended Reading

1. Chaudhuri S. and Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Rec., 26(1):65–74, 1997.
2. Kimball R. and Ross M. The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling. Wiley, New York, NY, USA, 2nd edn., 2002.

SOA

- ▶ [Service Oriented Architecture](#)

SOA Replication

- ▶ [Replication in Multi-Tier Architectures](#)

SOAP

ERIC WOHLSTADTER

University of British Columbia, Vancouver,
BC, Canada

Definition

SOAP [1] is an application-level protocol standard used to transport messages in distributed systems. The standard was defined and is maintained by the XML Protocol Working Group of the World Wide Web Consortium. SOAP is commonly used in the context of Web services. SOAP messages are encoded using XML and intended to carry XML encoded application data.

Key Points

SOAP provides a standard to separate infrastructure related data from application data for XML based messages. SOAP messages are known as “envelopes,” which contain both a header, for infrastructure data, and a body for application data. The infrastructure which handles messages for applications is referred to as a “SOAP node.” This role is commonly filled by some middleware platform. The SOAP protocol dictates the rules for the proper processing of messages by nodes on behalf of applications; this includes processing of header information and handling of faults.

The header processing rules are designed to make it easy to interpose network intermediaries between the sender and receiver of messages. The SOAP specification mentions that these intermediaries could be used for purposes such as “security services, annotation services, and content manipulation services.” The specification of header information used by specific kinds of intermediaries is left to other specifications commonly known as the WS-* proposals.

The SOAP specification is intended to be extensible so that different rules for message processing can be described in further specifications. These rules are called “message exchange patterns.” SOAP provides details of patterns for simple synchronous and asynchronous message exchange, which can be used for the purpose of remote procedure calls. The specification

mentions but does not provide details for other more stateful patterns such as conversational exchanges and peer-to-peer message routing.

When a SOAP node is unable to process a message, an error message, called a SOAP fault is issued. Several descriptive fault types are provided by the specification as well as the conditions under which each type should be used.

SOAP provides the foundation of a Web services stack. SOAP messages are commonly layered on top of the Hypertext Transfer Protocol (HTTP). This tends to make SOAP services easier to deploy behind network firewalls; although, some critics have argued this is an abuse of HTTP. Since XML messages tend to be much larger than their binary counterparts, SOAP provides guidelines for using a binary encoding of a SOAP message body.

SOAP was originally intended as the “Simple Object Access Protocol,” and its designers intended it to be used with traditional distributed object technologies such as remote method invocation. When SOAP became popular for Web services the acronym was dropped because Web service interfaces are agnostic as to whether object-oriented implementations are used.

Cross-references

- ▶ RMI
- ▶ Web Services
- ▶ W3C

Recommended Reading

1. SOAP Version 1.2, Part 1: Messaging Framework (2nd edn.). W3C Recommendation. <http://www.w3.org/TR/soap12-part1/>

Social Applications

MARISTELLA MATERA

Politecnico di Milano University, Milan, Italy

Synonyms

Web 2.0 applications; Collaborative software

Definition

Web applications, characteristic of the Web 2.0, that allow users to share data and interact with other users.

Key Points

The advent of the Web 2.0 has empowered the Web clients, thus providing users with richer and more complex interaction capabilities. The development of communication and interaction tools has therefore emerged, giving raise to computer-mediated communication and to collaborative approaches to content creation, based on new applications, such as social networking, file sharing, instant messaging, and blogs – just to mention few.

The advantage from the user experience perspective is that Web users do not play only the role of passive actors accessing information, but they become creators of the contents published by Web applications.

Very often, such new applications also foster the creation of *online communities*, i.e., groups of people that interact via Web-based communication media and cooperatively create contents.

Cross-references

- ▶ Visual Interaction
- ▶ Web 2.0/3.0

Social Networks

FELIX SCHWAGEREIT, STEFFEN STAAB

University of Koblenz-Landau, Koblenz, Germany

Definition

A social network is a social structure made of *actors*, which are discrete individual, corporate or collective social units like persons or departments [19] that are tied by one or more specific types of *relation* or interdependency, such as friendship, membership in the same organization, sending of messages, disease transmission, web links, airline routes, or trade relations. The actors of a social network can have other attributes, but the focus of the social network view is on the properties of the relational systems themselves [19]. For many applications social networks are treated as graphs, with actors as nodes and ties as edges. A *group* is the finite set of actors the ties and properties of whom are to be observed and analyzed. In order to define a group it is necessary to specify the network boundaries and the sampling. *Subgroups* consist of any subset of actors and the (possible) ties between them.

The science of social networks utilizes methods from general network theory and studies real world networks as well as structurally similar subjects dealing e.g., with information networks or biological networks.

Historical Background

The science of social network analysis comprises methods from social sciences, formal mathematical, statistical and computing methodology [19]. The first developments of scientific methods were empirically motivated and date back to the late nineteenth century. Jacob Moleno developed methods to facilitate the understanding of friendship patterns within small groups in the 1920s and 1930s. Other pioneers in the field of social networks were Davis, who studied social circles of women in an unnamed American city and Elton Mayo, who studied social networks of factory workers. Many of the current formal concepts (e.g., density, span, connectedness) had been introduced in the 1950s and 1960s as ways to describe social structures through measures. Another important milestone was an experiment Stanley Milgram conducted in 1967. In Milgram's experiment, a sample of US individuals were asked to reach a particular target person by passing a message along a chain of acquaintances. The average length of successful chains turned out to be about five intermediaries or six steps of separation.

Early research on social networks was limited to small networks with up to a few hundred actors, which could be examined visually. With increased computational power for data acquisition and management, networks may now comprise several millions of actors.

Foundations

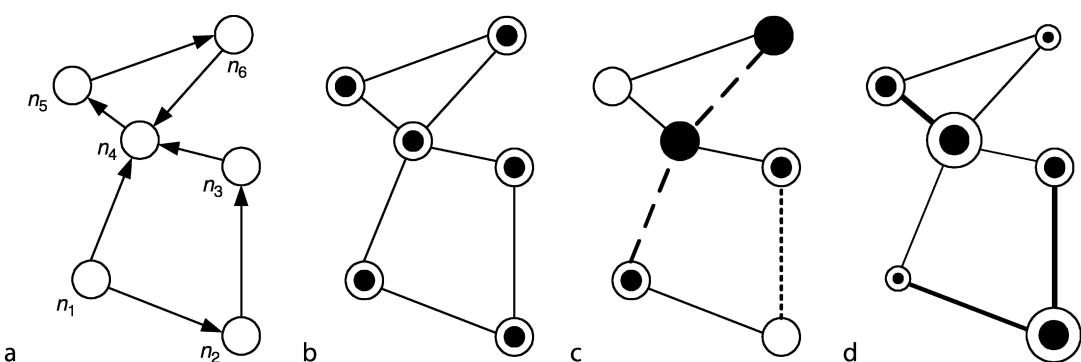
Types

The simplest type of network consists of only one set of actors and one relation representing one type of ties between the actors. More complex networks can be composed of different types of actors (*multi-mode*) and different relations (*multi-relational*). Furthermore the actors and ties between them can have assigned properties, which are mostly numerical. Ties can have a direction, which makes the network a directed graph. Figure 1 shows a selection of network types [12]. Network (i) is a directed network in which each edge has a direction; (ii) is an undirected network with only one type of actors; (iii) is a network with several types of actors and relations; (d) shows a network with different weights for actors and ties.

Of special interest in science of social networks are *bipartite graphs* [12] which contain actors of two types and ties connecting only actors of different types. They are called *affiliation networks* because they are suitable to express the membership of people (one type of actors) in groups (the second type of actors).

Notation

The common notation for social networks is the *socio-metric notation* [19]. Simple social networks with one relation and only one group of actors (like the one shown above) are represented as a matrix, called *sociomatrix* or *adjacency matrix*. For one relation X , let \mathbf{X} be the corresponding matrix. This matrix has g rows and g columns. The value at position x_{ij} denotes whether there exists a tie from the i th element of the social network to the j th element. An example sociomatrix for the social network (a) in Fig. 1 is shown in Table 1.



Social Networks. Figure 1. Types of social networks.

Social Networks. Table 1. Sociomatrix for network (a) in Fig. 1

	n_1	n_2	n_3	n_4	n_5	n_6
n_1	-	1	0	1	0	0
n_2	0	-	1	0	0	0
n_3	0	0	-	1	0	0
n_4	0	0	0	-	1	0
n_5	0	0	0	0	-	1
n_6	0	0	0	1	0	-

For more complex networks, like multi-mode and/or multi-relational social networks, tensors may be used instead of matrices [18].

Measures

Measures have been developed in order to formalize local and global properties for social networks. Local and global properties of social networks describe ego-centric properties of individual actors and socio-centric properties of the network as a whole, respectively. Furthermore, subsets of actors (subgroups) can be determined. The following paragraphs contain an outline of several basic concepts.

Socio-centric Properties In order to compare different social networks in size and structure the following basic measures have been established.

- *Number of Actors:* g
- *Number of Ties:* m
- *Mean Standardized Degree (Density):* $z = \frac{\sum C_D(n_i)}{g(g-1)}$
- *Mean Actor-Actor Distance / Characteristic Path Length:* $l = \frac{1}{\frac{1}{2}g(g+1)} \sum_{i,j} d(n_i, n_j)$
- *Diameter:* is the longest Distance between all pairs of nodes of a given network. The distance $d(n_i, n_j)$ between a pair of nodes n_i and n_j in the network is the length of the geodesic (which is the shortest path between the two nodes).

Ego-centric Properties The identification of the “most important” or “prominent” actor was one of the primary goals of social network analysis [19]. Therefore various measures were developed to quantify “importance” of actors and subgroups for a given social network. The following measures can be calculated for simple undirected graphs of social networks.

- *Actor Degree Centrality* is the count of the number of ties to other actors in the network. The relevance of

this measure is based on the assumption that an actor, which has more connections than other actors can be considered more active and therefore important. The actor degree centrality is calculated from sociomatrix X as follows:

$$C_D(n_i) = \sum_j x_{ij}$$

- *Actor Closeness Centrality* is the degree to which an individual is close to all other individuals in a network (directly or indirectly). Therefore an actor is central if it can quickly (that means by relying on so few mediators as possible) interact with all other actors. The index of actor closeness-centrality is:

$$C_C(n_i) = \left[\sum_{j=1, j \neq i}^g d(n_i, n_j) \right]^{-1}$$

where $d(n_i, n_j)$ is the length of the geodesic of actor i and actor j . To allow comparisons between different networks actor closeness can be standardized:

$$C'_C(n_i) = \frac{g-1}{\left[\sum_{j=1, j \neq i}^g d(n_i, n_j) \right]}$$

- *Actor Betweenness Centrality* is the degree to which an individual lies between other individuals in the network. Therefore it is based on the assumption that all other actors lying in between have a certain amount of control on the interaction relying on them. So the betweenness of an actor is higher if more of the possible interactions rely on it as mediator. In order to calculate betweenness centrality two other measures are needed: g_{jk} , the number of geodesics linking two actors j and k ; as well as $g_{ik}(n_i)$, which is the number of geodesics linking two actors that contain the actor i :

$$C_B(n_i) = \sum_{j < k} g_{jk}(n_i) / g_{ik}$$

For comparisons the measure can be normalized:

$$C'_B(n_i) = C_B(n_i) / [(g-1)(g-2)/2]$$

Subgroups In most social networks actors organize themselves in subgroups or cliques, which have their

own values, sub-cultures, and structures. Therefore several methods to define and recognize certain kinds of subgroups were developed [19].

- A *Clique of size k* is a subgroup consisting of k many actors which are all adjacent to each other.
- An *n-Clique* is a subgroup with the property that the distance (length of the geodesic) between all actors is no greater than n and there is no actor with a distance equal or less than n outside the n-clique. An n-clique with $n = 1$ is equal to a normal clique.
- A *k-Core* is a subgroup with each actor is adjacent to at least n other actors in the subgroup.
- A *Cluster* is a subgroup consisting of actors which are similar to each other. The similarity (structural equivalence) of two actors can be defined with criteria like euclidean distance or correlation based on vectors of a sociomatrix. Similarity based clusters in undirected networks are usually created by using agglomerative or divisive hierarchical clustering methods [14]. For clustering directed networks methods like directed spectral clustering [7] can be used. In general graph theory there exist methods for partitioning graphs which can also be applied to graphs of social networks. One of these methods is the min-max cut algorithm which pursues the goal of minimizing the similarity between subgraphs while maximizing the similarity within each subgraph. Other clustering approaches are based on methods for finding densely connected subgroups by the calculation of special clustering coefficients or by comparing the number of connections within a subgroup with the number of connections to outside actors.

Topological Properties

Small-World Topology The small-world model [20] is a well studied distribution model of actors and ties, since it has interesting properties and features. Due to the fact that networks often have a geographical component to them it is reasonable to assume that geographical proximity will play a role in deciding which actors are connected. So in a small-world network each actor is connected to actors in its near neighborhood. Other connections between more distant actors (long-range connections) are infrequent and have a low probability. The probability for each actor of having a degree k follows a power law $p_k \sim k^{-\alpha}$ with α as

constant scaling exponent. Despite the fact that long-range connections occur only sporadically the diameter of small-world networks is exponentially smaller than their size, being bounded by a polynomial in $\log g$, where g is the number of nodes. In other words, there is always a very short path between any two nodes [8].

The discovery that real world social networks might have small-world characteristics explains the importance of this model. So it can be observed that the chain of social acquaintances required to connect one arbitrary person to another arbitrary person anywhere in the world is generally short. This concept gave rise to the famous phrase six degrees of separation after a 1967 small-world experiment by Stanley Milgram. Academic researchers continue to explore this phenomenon. A recent electronic small-world experiment [5] at Columbia University showed that about five to seven degrees of separation are sufficient for connecting any two people through e-mail. Other applications of the small-world model are investigations of iterated games, diffusion processes or epidemic processes [12].

Creation of Networks

Artificially generated graphs allow comparison with real datasets and by analyzing and comparing their properties they give insights into the inner structure of social networks. They also allow for the generation of (overlay) network structures on top of existing information structures.

Several procedures are known to generate social networks from scratch. A *Poisson random graph* is the simplest way to construct a social network. This is simply done by connecting each pair of actors with the probability of p . The result of this procedure is a network with a Poisson degree distribution ($p_k = \frac{\lambda^k}{k!} e^{-\lambda}$). Since this distribution is unlike the highly skewed power-law distributions of real world networks other methods have been proposed [12].

One of the important methods is known as *preferential attachment* [1]. In this model, new nodes are added to a pre-existing network, and connected to each of the original nodes with a probability proportional to the number of connections each of the original nodes already had. I.e., new nodes are more likely to attach to hubs than peripheral nodes or in other words the “rich-get-richer”. Statistically, this method will generate a power-law distributed small-world network (that is, a scale-free network).

Since there is evidence that the preferential attachment model does not show all the properties real world networks obey, like increasing of the average degree and shrinking of the diameter on growing of a network, other models have been proposed [9]. The *Community Guided Attachment*, which is based on a decomposition of actors into a nested set of subgroups, such that the difficulty of forming new links between subgroups increases with the size of the subgroups. In the *Forest Fire Model* new actors are attached to the network by burning through existing ties in epidemic fashion.

Key Applications

Distributed Information Management

Social routing allows to route efficiently in peer-to-peer networks without knowledge about the global network structure. This routing with local knowledge can be achieved by regarding the network as a social network and exploiting several properties of social networks like small-world characteristics [8,10].

Information Replication in information networks can improve scalability and reliability. By performing social network clustering on these structures prefetching of content can be improved [15].

Information Extraction

Name disambiguation is a technique for distinguishing person names in unsupervised information frameworks (e.g., web pages), where unique identifiers can not be assumed [2].

Ontology Extraction methods can be performed on social network structures like communities and their folksonomies. This approach is based on the assumption that individual interactions of a large number of actors might lead to global effects that could be observed as semantics [11].

Social Recommendations

Social networking portals like Xing or LinkedIn allow users to express their relationships to other users and to provide personal information. This social network can be used e.g., for finding a short path to persons in special positions by identifying the geodesic to them [16].

Filtering, recommendations and inferred trust can be improved by taking into account the social networks all relevant actors are involved. So e.g., the

trustworthiness of Bob can be inferred from a social network by Alice even if both are not directly known to each other [6].

Viral marketing is the strategy to let satisfied customers distribute advertisements (e.g., video clips) by recommendation or forwarding to other potential customers they know. Viral marketing campaigns are usually started by sending the advertisements to actors holding central positions in social networks in order to facilitate a rapid distribution [13,17].

Future Directions

For the future of the social network science many areas remain insufficiently explored [12]. Many properties of social networks have been studied in the past decades. But the scientific community is still lacking the whole picture which shows what the most important properties for each application are. Especially generalized propositions (e.g., “Are more centralized organizations more efficient?”) about the structure of social networks need further verification across a large number of networks [19]. Another important direction of future research is to improve the understanding of the dynamics in and the evolution of social networks [3]. In order to archive this new and more sophisticated models of social networks have to be developed. New kinds of data including more complex structures and new properties of actors or relations demand further generalization of current models. An example of these more complex structures are multiple relations which connect more than two actors.

Data Sets

- Enron Email dataset (<http://www.cs.cmu.edu/~enron/> and <http://www.enronemail.com/>) contains about 600,000 Email messages belonging to 156 users. It was made public during the legal investigation concerning the Enron corporation.
- The Internet Movie Data Base (IMDB) (<http://www.imdb.com/interfaces/>) is a collection of data about movies (about 400,000) and actors (about 900,000). Especially the affiliation network of the co-appearance of actors in the same movie is subject of several studies. (cf. “The Oracle of Bacon” <http://oracleofbacon.org/>)
- Digital Bibliography & Library Project (DBPL) collects the bibliographic information on major computer science journals and proceedings (currently

about 950,000 articles). Similar to the IMDB the co-authorship can be used to generate affiliation networks. (dataset <http://dblp.uni-trier.de/xml/>)

- Southern Woman Dataset, which was collected in the 1930s is published in the classical study of Davis [4], a pioneer of social network analysis. It contains the attendance at 14 social events by 18 women in an unnamed US city.

URL to Code

Tools and Libraries

(cf. <http://www.insna.org/software/index.html>):

- Jung: <http://jung.sourceforge.net/>
- Pajek: <http://vlado.fmf.uni-lj.si/pub/networks/pajek/default.htm>
- UCINET: <http://www.analytictech.com/ucinet/uci.net.htm>

Conference Series

- International Sunbelt Social Network Conferences: <http://www.insna.org/sunbelt/index.html>

Journals

- Social Networks: <http://www.innsa.org/pubs/connections/index.html>
- CONNECTIONS: <http://www.insna.org/indexConnect.html>
- Journal of Social Structure: <http://www.cmu.edu/joss/>

Cross-references

- ▶ Biological Networks
- ▶ Cluster and Distance Measure
- ▶ Clustering Overview and Applications
- ▶ Graph
- ▶ Hierachial Clustering
- ▶ Web Characteristics and Evolution

Recommended Reading

1. Barabási A.L. and Albert R. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.
2. Bekkerman R. and McCallum A. Disambiguating Web appearances of people in a social network. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 463–470.
3. Berners-Lee T., Hall W., Hendler J., Shadbolt N., and Weitzner D.J. Creating a science of the Web. *Science*, 313:769–771, 2006.

4. Davis A., Gardner B.B., and Gardner M.R. *Deep South*. The University of Chicago Press, 1941.
5. Dodds P.S., Muhamad R., and Watts D. An experimental study of search in global social networks. *Science*, 301:827–829, 2003.
6. Golbeck J. and Hendler J.A. Inferring binary trust relationships in Web-based social networks. *ACM Trans. Internet Techn.*, 6(4):497–529, 2006.
7. Huang J., Zhu T., and Schuurmans D. Web communities identification from random walks. In Proc. Joint European Conf. on Machine Learning and European Conference on Principles and Practice of Knowledge Discovery in Databases, 2006.
8. Kleinberg J. Navigation in a small world. *Nature*, 406:845, 2000.
9. Leskovec J., Kleinberg J., and Faloutsos C. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1):2, 2007.
10. Löser A., Staab S., and Tempich C. Semantic Social Overlay Networks. *IEEE Journal on Selected Areas in Communication*, 25(1):5–14, 2007.
11. Mika P. *Social Networks and the Semantic Web*. Springer, 2007.
12. Newman M.E.J. The Structure and Function of Complex networks. *SIAM Rev.*, 45(2):167–256, 2003.
13. Richardson M. and Domingos P. Mining knowledge-sharing sites for viral marketing. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 61–70.
14. Scott J. *Social Network Analysis: A Handbook*. Sage, 2000.
15. Sidiropoulos A., Pallis G., Katsaros D., Stamos K., Vakali A., and Manolopoulos Y. Prefetching in content distribution networks via web communities identification and outsourcing. *World Wide Web J.*, 11(1):39–70, 2008.
16. Staab S., Domingos P., Mika P., Golbeck J., Ding L., Finin T.W., Joshi A., Nowak A., and Vallacher R.R. Social networks applied. *IEEE Intell. Syst.*, 20(1):80–93, 2005.
17. Subramani M.R. and Rajagopalan B. Knowledge-sharing and influence in online social networks via viral marketing. *Commun. ACM*, 46(12):300–307, 2003.
18. Sun J., Tao D., and Faloutsos C. Beyond streams and graphs: dynamic tensor analysis. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006, pp. 374–383.
19. Wasserman S. and Faust K. *Social network analysis*. Cambridge University Press, Cambridge, 1994.
20. Watts D.J. and Strogatz S.H. Collective dynamics of “small-world” networks. *Nature*, 393:440–442, 1998.

Software Transactional Memory

KEIR FRASER

University of Cambridge, Cambridge, UK

Definition

Software transactional memory (STM) is a method of concurrency control in which shared-memory accesses are grouped into transactions which either

succeed or fail to commit in their entirety. STM provides applications programmers with an alternative to mutual-exclusion locks which avoids many of the latter's pitfalls, including risk of deadlock, unnecessary serialization, and priority inversion. Many STMs are themselves implemented using lock-free programming methods, although this is not a hard-and-fast rule.

Key Points

A software transactional memory (STM) is a software library or programming-language feature which provides application programmers with an interface for allocating and accessing shared-memory variables [3]. These variables are accessible in a concurrency-safe manner without resorting to classical concurrency-management techniques such as mutual exclusion. This is achieved by grouping accesses into transactions which execute in isolation and then atomically succeed or fail in their entirety.

The application programmer chooses when transactions should start and end, rather like choosing when to acquire and release mutexes in a conventional multi-threaded program, to ensure consistency of application data structures. The STM implementation is responsible for ensuring that transactions execute in isolation and coomit atomically. Thus transactional memory guarantees the same ACID properties as classical database transactions, with the exception of durability.

The benefits of a transactional interface to shared memory are numerous. Traditional mutexes, when used conservatively, can lead to unnecessary serialization of operations that do not otherwise conflict. When a finer-grained approach is taken, involving multiple locks with individually smaller scope, the programmer must take care to avoid subtle deadlock scenarios. STM is perhaps the most promising of the proposed lock-free techniques which eschew traditional mutual exclusion and hope to enable the average programmer to implement scalable multi-threaded applications in mainstream languages [1]. Hence, although still in its infancy and an ongoing topic of research, STM is being viewed eagerly by an industry looking for salvation from the complexity of optimizing for modern multi-core systems [2].

Cross-references

- ▶ Performance Analysis
- ▶ Transaction

Recommended Reading

1. Fraser K. and Harris T. Concurrent programming without locks. *ACM Trans. Comput. Syst.*, 25(2), 2007.
2. Saha B., Adl-Tabatabai A., Hudson R., Minh C., and Hertzberg B. McRT-STM: a high performance software transactional memory system for a multi-core runtime. In Proc. 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 2006, pp. 187–197.
3. Shavit N. and Touitou D. Software transactional memory. In Proc. ACM SIGACT-SIGOPS 14th Symp. on the Principles of Dist. Comp., 1995, pp. 204–213.

SONs

- ▶ Semantic Overlay Networks
-

Sort-Merge Join

JINGREN ZHOU

Microsoft Research, Redmond, WA, USA

Synonyms

Merge join

Definition

The sort-merge join is a common join algorithm in database systems using sorting. The join predicate needs to be an equality join predicate. The algorithm sorts both relations on the join attribute and then merges the sorted relations by scanning them sequentially and looking for qualifying tuples.

Key Points

The sorting step groups all tuples with the same value in the join attribute together. Such groups are sorted based on the value in the join attribute so that it is easy to locate groups from the two relations with the same attribute value. Sorting operation can be fairly expensive. If the size of the relation is larger than the available memory, external sorting algorithm is required. However, if one input relation is already clustered (sorted) on the join attribute, sorting can be completely avoided. That is why the sort-merge join looks attractive if any of the input relations is sorted on the join attribute.

The merging step starts with scanning the relations R and S and looking for matching groups from the two relations with the same attribute value. The two scans

start at the first tuple in each relation. The algorithm advances the scan of R as long as the current R tuple has an attribute value which is less than that of the current S tuple. Similarly, the algorithm advances the scan of S as long as the current S tuple has an attribute value which is less than that of the current R tuple. The algorithm alternates between such advances until an R tuple \mathcal{R} and an S tuple \mathcal{S} with $\mathcal{R}.r = \mathcal{S}.s$. The join tuple $\{\mathcal{R}, \mathcal{S}\}$ is added to result.

There could be several R tuples and several S tuples with the same attribute value as the current tuples \mathcal{R} and \mathcal{S} . That is, several R tuples may belong to the current \mathcal{R} group since they all have the same attribute value. The same applies to the current \mathcal{S} group. Every tuple in the current \mathcal{R} group joins with every tuple in the current \mathcal{S} group. The algorithm then resumes scanning R and S , beginning with the first tuples that follow the group of tuples that are just processed.

When the two relations are too large to be held in available memory, one improvement is to combine the merging step of external sorting with the merging step of the join if the number of buffers available is larger than the total number of sorted runs for both R and S . The idea is to allocate one buffer page for each run of R and one for each run of S . The algorithm merges the runs of R , merges the runs of S , and joins (merges) the resulting R and S streams as they are generated.

```
Algorithm 1: Sort-Merge Join:  $R \bowtie_{R.r=S.s} S$ 
// sorting step
Sort the relation  $R$  on the attribute  $r$ ;
Sort the relation  $S$  on the attribute  $s$ ;
// merging step
 $\mathcal{R}$  = first tuple in  $R$ ;
 $\mathcal{S}$  = first tuple in  $S$ ;
 $\mathcal{S}'$  = first tuple in  $S$ ;
while  $\mathcal{R} \neq \text{eof}$  and  $\mathcal{S}' \neq \text{eof}$  do
    while  $\mathcal{R}.r < \mathcal{S}'.s$  do
        |  $\mathcal{R}$  = next tuple in  $R$  after  $\mathcal{R}$ ;
    end
    while  $\mathcal{R}.r > \mathcal{S}'.s$  do
        |  $\mathcal{S}'$  = next tuple in  $S$  after  $\mathcal{S}'$ ;
    end
     $\mathcal{S} = \mathcal{S}'$ ;
    while  $\mathcal{R}.r == \mathcal{S}.s$  do
        |  $\mathcal{S} = \mathcal{S}'$ ;
        | while  $\mathcal{R}.r == \mathcal{S}.s$  do
            | | add  $\{\mathcal{R}, \mathcal{S}\}$  to result;
            | |  $\mathcal{S} = \text{next tuple in } S \text{ after } \mathcal{S}$ ;
        end
        |  $\mathcal{R}$  = next tuple in  $R$  after  $\mathcal{R}$ ;
    end
     $\mathcal{S}' = \mathcal{S}$ ;
end
```

Cross-references

- ▶ [Evaluation of Relational Operators](#)
- ▶ [External Sorting](#)
- ▶ [Parallel Join Algorithms](#)

Recommended Reading

1. Mishra P. and Eich M.H. Join processing in relational databases. *ACM Comput. Surv.*, 24(1):63–113, 1992.

Source

- ▶ [Provenance](#)
- ▶ [Provenance in Scientific Databases](#)

Space-Filling Curves

MOHAMED F. MOKBEL¹, WALID G. AREF²

¹University of Minnesota, Minneapolis, MN, USA

²Purdue University, West Lafayette, IN, USA

Synonyms

Distance-preserving mapping; Locality-preserving mapping; Multi-dimensional mapping; Linearization

Definition

A space-filling curve (SFC) is a way of mapping the multi-dimensional space into the one-dimensional space. It acts like a thread that passes through every cell element (or pixel) in the multi-dimensional space so that every cell is visited exactly once. Thus, a space-filling curve imposes a linear order of points in the multi-dimensional space. A D -dimensional space-filling curve in a space of N cells (pixels) of each dimension consists of $N^D - 1$ segments where each segment connects two consecutive D -dimensional points. There are numerous kinds of space-filling curves (e.g., Hilbert, Peano, and Gray). The difference between such curves is in their way of mapping to the one-dimensional space, i.e., the order that a certain space-filling curve traverses the multi-dimensional space. The quality of a space-filling curve is measured by its ability in preserving the locality (or relative distance) of multi-dimensional points in the mapped one-dimensional space. The main idea is that any two D -dimensional points that are close by in the

D -dimensional space should be also close by in the one-dimensional space.

Key Points

Space-filling curves are discovered by Peano [3] where he introduces a mapping from the unit interval to the unit square. Hilbert [1] generalizes the idea to a mapping of the whole space. Following Peano and Hilbert curves, many space-filling curves are proposed, e.g., [4]. Space-filling curves are classified into two categories: recursive space-filling curves (RSFC) and non-recursive space-filling curves. An RSFC is an SFC that can be recursively divided into four square RSFCs of equal size. Examples of RSFCs are the Peano SFC, the Gray SFC, and the Hilbert SFC. For the past two decades, recursive space-filling curves have been considered a natural method for locality-preserving mappings. Recursive space-filling curves are special case of fractals [2]. Mandelbrot [2], the father of fractals, derived the term fractal from the Latin adjective fractus. The corresponding Latin verb frangere means “to break” or “to fragment.” Thus, fractals divide the space into a number of fragments, visiting the fragments in a specific order. Once a fractal starts to visit points from a certain fragment, no other fragment is visited until the current one is completely exhausted. By dealing with one fragment at a time, fractal locality-preserving mapping algorithms perform a local optimization based on the current fragment.

Cross-references

- ▶ High-Dimensional Indexing
- ▶ Space Filling Curves for Query Processing
- ▶ Spatial Indexing Techniques

Recommended Reading

1. Hilbert D. Ueber stetige abbildung einer linie auf ein flächenstück. *Math. Ann.*, 459–460, 1891.
2. Mandelbrot B.B. *Fractal geometry of nature*. W. H. Freeman, New York, 1977.
3. Peano G. Sur une courbe qui remplit toute une aire plaine. *Math. Ann.*, 36:157–160, 1890.
4. Sagan H. *Space Filling Curves*. Springer, Berlin Heidelberg New York, 1994.

Space Partitioning

- ▶ Indexing and Similarity Search

Space Segmentation

- ▶ Indexing and Similarity Search

Space-Filling Curve

- ▶ Fractal

Space-Filling Curves for Query Processing

MOHAMED F. MOKBEL¹, WALID G. AREF²

¹University of Minnesota, Minneapolis, MN, USA

²Purdue University, West Lafayette, IN, USA

Synonyms

Distance-preserving mapping; Locality-preserving mapping; Multi-dimensional mapping; Linearization

Definition

Given a query Q , a one-dimensional index structure I (e.g., B-tree), and a set of D dimensional points, a space-filling curve S is used to map the D dimensional points into a set of one-dimensional points that can be indexed through I for an efficient execution of query Q . The main idea is that space-filling curves are used as a way of mapping the multi-dimensional space into the one-dimensional space such that existing one-dimensional query processing and indexing techniques can be applied.

Historical Background

Although space-filling curves were discovered in 1890 [14], their use in query processors has emerged only in the last two decades as it is mainly motivated by the emergence of multi-dimensional applications. In particular, space-filling curves have been used as a mapping scheme that supports spatial join algorithms [13], spatial access methods [2,7], efficient processing of range queries [1,6], and nearest-neighbor queries in [8]. Numerous algorithms are developed for efficiently generating different space-filling curves that include recursive

algorithms for the Hilbert SFC [4,15], recursive algorithms for the Peano SFC [15], table-driven algorithms for the Peano and Hilbert SFCs [4]. The clustering and mapping properties of various space-filling curves have been extensively studied in the literature (e.g., see [10,12]).

Foundations

Mapping Scheme

Figures 1 and 2 give examples of two- and three-dimensional space-filling curves with grid size (i.e., number of points per dimension) eight and four, respectively. Space-filling curves are classified into two categories: *recursive* space-filling curves (RSFC) and *non-recursive* space-filling curves. An RSFC is an SFC that can be recursively divided into four square RSFCs of equal size. Non-recursive space-filling curves include the Sweep SFC (Figs. 1a and 2a), the Scan SFC (Figs. 1b and 2b), the Diagonal SFC (Fig. 1f), and the Spiral SFC (Fig. 1g). Recursive space-filling curves include the Peano SFC (Figs. 1c and 2c), the Gray SFC (Figs. 1d and 2d), and the Hilbert SFC (Figs. 1e

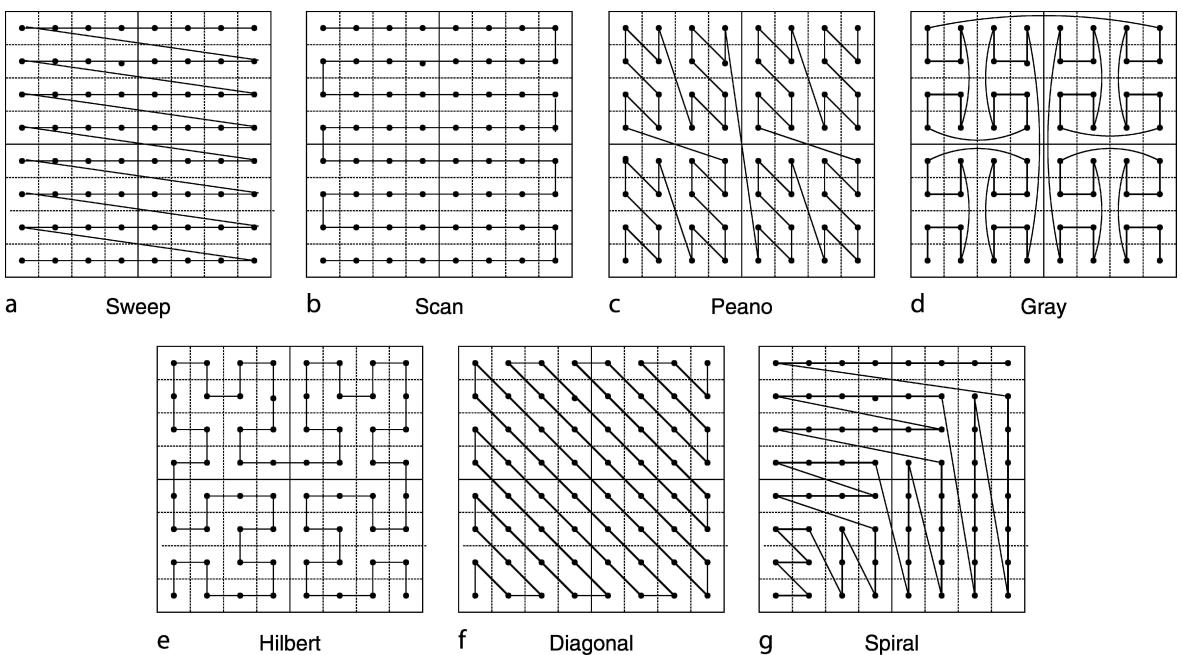
and 2e). Table 1 gives the first 16 visited points for the Peano, Gray, and Hilbert space-filling curves.

The Peano SFC

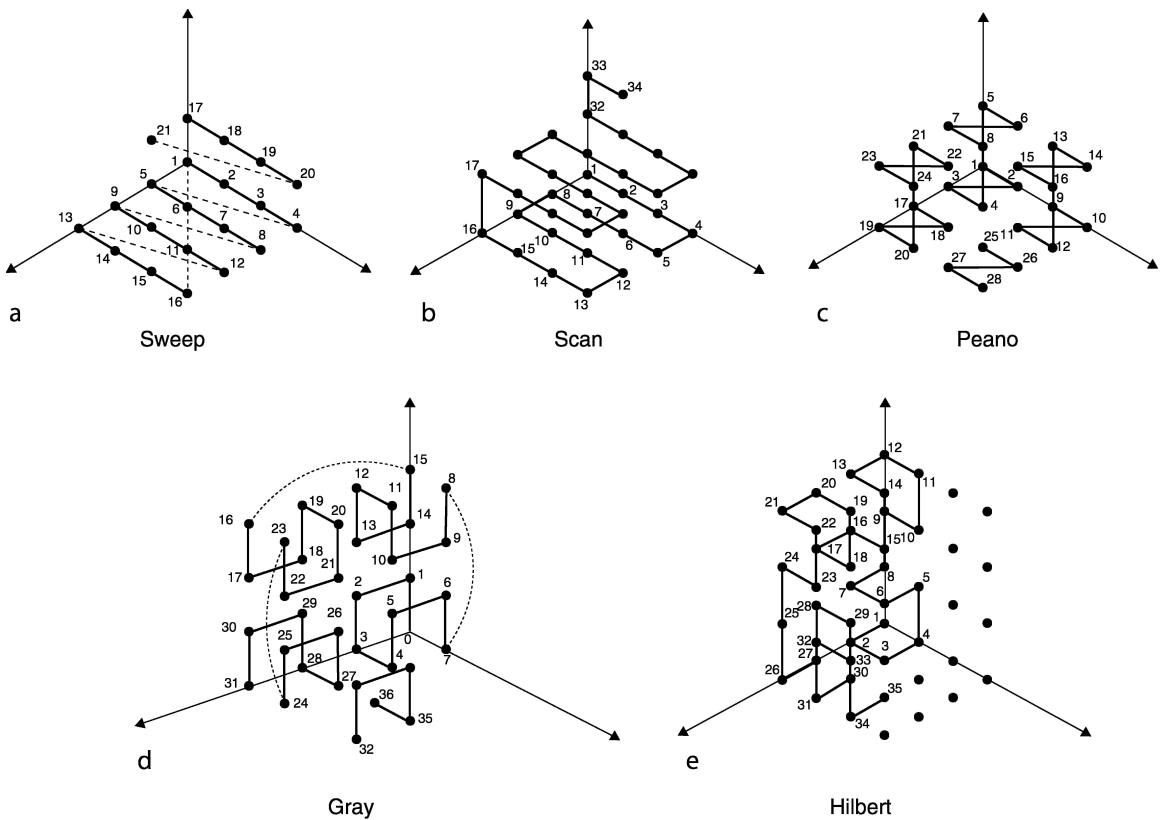
The Peano SFC (Figs. 1c and 2c) is introduced by Peano [14] and is also termed Morton encoding, quad code, bit-interleaving, N-order, locational code, or Z-order. The Peano SFC is constructed recursively as in Fig. 3. The basic shape (Fig. 3a) contains four points in the four quadrants of the space. Each quadrant is represented by two binary digits. The most significant digit is represented by its x position while the least significant digit is represented by its y position. The Peano SFC orders space quadrants in ascending order (00, 01, 10, 11). Figure 3b contains four blocks of Fig. 3a at a finer resolution and is visited in the same order as in Fig. 3a. Similarly, Fig. 3c contains four blocks of Fig. 3b at a finer resolution.

The Gray SFC

The Gray SFC (Figs. 1d and 2d) uses the Gray code representation [5] in contrast to the binary code representation as in the Peano SFC. Figure 4 gives the



Space-Filling Curves for Query Processing. Figure 1. Two-dimensional space-filling curves.



Space-Filling Curves for Query Processing. Figure 2. Three-dimensional space-filling curves.

recursive construction of the Gray SFC. The basic shape (Fig. 4a) contains four points in the four quadrants of the space. The Gray SFC visits the space quadrants in ascending order according to the Gray code (00, 01, 11, 10). Figure 4b is constructed by having the first and fourth blocks as those of Fig. 4a, while the second and the third blocks are the rotation of the blocks in Fig. 4a by 180^0 . Similarly, Fig. 4c is constructed from two blocks of Fig. 4b at a finer resolution and two blocks of the rotation of Fig. 4b by 180^0 .

The Hilbert SFC

Figure 5 gives the recursive construction of the Hilbert SFC. The basic block of the Hilbert SFC (Fig. 5a) is the same as that of the Gray SFC (Fig. 4a). The basic block is repeated four times at a finer resolution in the four quadrants, as given in Fig. 5b. The quadrants are visited in their gray order. The second and third blocks in Fig. 5b have the same orientation as in Fig. 5a. The first block is constructed from rotating the block of Fig. 5a

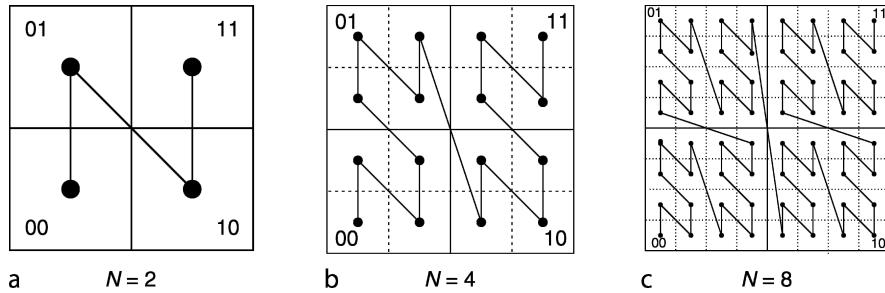
by 90^0 , while the fourth block is constructed by rotating the block of Fig. 5 by -90^0 . Similarly, Fig. 5a is constructed from Fig. 5b.

Segment Types

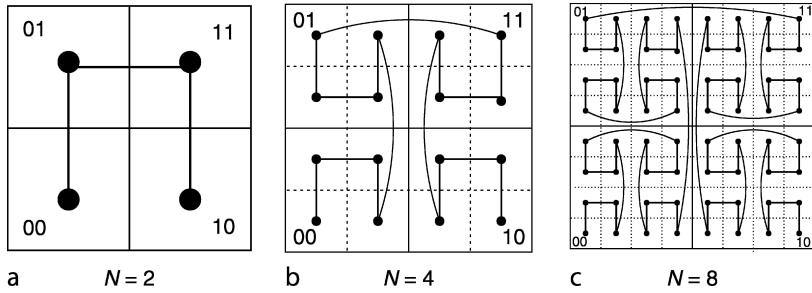
A space-filling curve consists of a set of segments. Each segment connects two consecutive multi-dimensional points. Five different types of segments are distinguished, namely, *Jump*, *Contiguity*, *Reverse*, *Forward*, and *Still*. A *Jump* segment in an SFC is said to happen when the distance, along any of the dimensions, between two consecutive points in the SFC is greater than one. Similarly, a *Contiguity* segment in an SFC is said to happen when the distance, along any of the dimensions, between two consecutive points in the SFC is equal to one. On the other side, a segment in an SFC is termed a *Reverse* segment if the projection of its two consecutive points, along any of the dimensions, results in scanning the dimension in decreasing order. Similarly, a segment in an SFC is termed a *Forward* segment if the projection of its two consecutive points,

Space-Filling Curves for Query Processing. Table 1. The first 16 traversed points by two-dimensional Peano, Gray, and Hilbert space-filling curves

Point	Peano	Gray	Hilbert	Point	Peano	Gray	Hilbert
0	(0,0)	(0,0)	(0,0)	8	(2,0)	(3,3)	(2,2)
1	(0,1)	(0,1)	(0,1)	9	(2,1)	(3,2)	(3,2)
2	(1,0)	(1,1)	(1,1)	10	(3,0)	(2,2)	(3,3)
3	(1,1)	(1,0)	(1,0)	11	(3,1)	(2,3)	(2,3)
4	(0,2)	(1,3)	(2,0)	12	(2,2)	(2,0)	(1,3)
5	(0,3)	(1,2)	(3,0)	13	(2,3)	(2,1)	(1,2)
6	(1,2)	(0,2)	(3,1)	14	(3,2)	(3,1)	(0,2)
7	(1,3)	(0,3)	(2,1)	15	(3,3)	(3,0)	(0,3)



Space-Filling Curves for Query Processing. Figure 3. The Peano SFC.



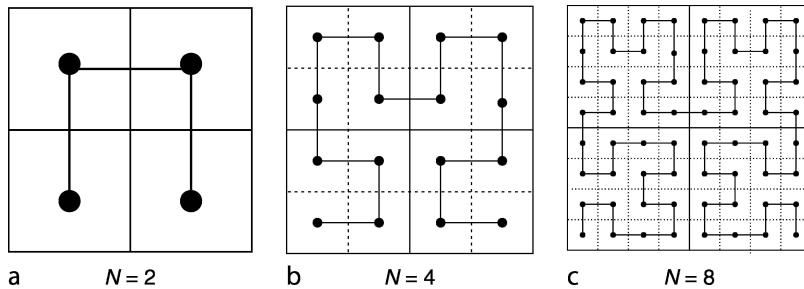
Space-Filling Curves for Query Processing. Figure 4. The Gray SFC.

along any of the dimensions, results in scanning the dimension in increasing order. Finally, a segment in an SFC is termed a *Still* segment when the distance, along any of the dimensions, between the segment's two consecutive points in the SFC is equal to zero. Closed formulas to count the number of *Jump*, *Contiguity*, *Reverse*, *Forward*, and *Still* segments along each dimension can be found in [10].

Irregularity

An optimal locality-preserving space-filling curve is one that sorts multi-dimensional points in ascending

order for all dimensions. However, in reality, when a space-filling curve attempts to sort the points in ascending order according to one dimension, it fails to do the same for the other dimensions. A good space-filling curve for one dimension is not necessarily good for the other dimensions. In order to measure the mapping quality of a space-filling curve, the concept of *irregularity* has been introduced as a measure of goodness for the order imposed by a space-filling curve [11]. Irregularity introduces a quantitative measure that indicates the non-avoidable reverse order imposed by space-filling curves for some or all



Space-Filling Curves for Query Processing. Figure 5. The Hilbert SFC.

dimensions. Irregularity is measured for each dimension separately, and gives an indicator of how a space-filling curve is far from the optimal. The lower the irregularity, the better the space-filling curve. The irregularity is formally defined as: For any two points, say P_i and P_j , in the D -dimensional space with coordinates $(P_i.u_0, P_i.u_1, \dots, P_i.u_{D-1})$, $(P_j.u_0, P_j.u_1, \dots, P_j.u_{D-1})$, respectively, and for a given space-filling curve S , if S visits P_i before P_j , an irregularity occurs between P_i and P_j in dimension k iff $P_j.u_k < P_i.u_k$. Closed formulas to count the number of irregularities for various space-filling curves can be found in [11].

Key Applications

Pre-processing for Multi-dimensional Applications: Multimedia Databases, GIS, and Multi-dimensional Indexing

Mapping the multi-dimensional space into the one-dimensional domain plays an important role in applications that involve multi-dimensional data. Multimedia databases, Geographic Information Systems (GIS), QoS routing, and image processing are examples of multi-dimensional applications. Modules that are commonly used in multi-dimensional applications include searching, sorting, scheduling, spatial access methods, indexing, and clustering. Considerable research has been conducted for developing efficient algorithms and data structures for these modules for one-dimensional data. In most cases, modifying the existing one-dimensional algorithms and data structures to deal with multi-dimensional data results in spaghetti-like programs to handle many special cases. The cost of maintaining and developing such code degrades the system performance. Mapping from the multi-dimensional space into the one-dimensional domain provides a pre-processing step for multi-dimensional applications. The pre-processing step takes the multi-dimensional data

as input and outputs the same set of data represented in the one-dimensional domain. The idea is to keep the existing algorithms and data structures independent of the dimensionality of data. The objective of the mapping is to represent a point from the D -dimensional space by a single integer value that reflects the various dimensions of the original space. Such a mapping is called a locality-preserving mapping in the sense that, if two points are near to each other in the D -dimensional space, then they will be near to each other in the one-dimensional space.

Network-Attached Storage Devices NASDs

Writing efficient schedulers is becoming a very challenging task, given the increase in demand of such systems. Consider the case of network-attached storage devices (NASDs) [3] as a building block for a multimedia server. NASDs are smart disks that are attached directly to the network. In a multimedia server, a major part of a NASD function goes towards fulfilling the real-time requests of users. This involves disk and network scheduling with real-time constraints, possibly with additional requirements like request priorities, and quality-of-service guarantees. NASDs requirements can be mapped in the multi-dimensional space and a SFC-based scheduler is used. The type of space-filling curve used in NASD scheduling is determined by its requirements. For example, in NASD, if reducing the number of requests that lose their deadlines is more important than increasing the disk or network bandwidth, then the real-time deadline dimension of the scheduling space will be favored. As a result, a space-filling curve with intentional bias is favored.

Multimedia Disk Scheduling

Consider the problem of disk scheduling in multimedia servers [9]. In addition to maximizing the bandwidth of the disk, the scheduler has to take into

consideration the real-time constraints of the page requests, e.g., as in the case of video streaming. If clients are prioritized based on quality-of-service guarantees, then the disk scheduler might as well consider the priority of the requests in its disk queue. Writing a disk scheduler that handles real-time and QoS constraints in addition to maximizing the disk bandwidth is challenging and a hard task. Scheduler parameters can be mapped to space dimensions and an SFC-based scheduler is used. The reader is referred to [9] to get more insight about the applicability of the irregularity in multi-media disk schedulers.

Future Directions

Future directions for space-filling curves include: (i) exploiting new multi-dimensional applications that can make use of the properties of space-filling curves, (ii) analyzing the behavior of various space-filling curves in high-dimensional space, (iii) providing automated modules with the ability of choosing the appropriate space-filling curve for a given application, and (iv) developing new space-filling curves that are tailored to specific applications.

Cross-references

- ▶ [High-Dimensional Indexing](#)
- ▶ [Space-Filling Curves](#)
- ▶ [Spatial Indexing Techniques](#)

Recommended Reading

1. Faloutsos C. Gray codes for partial match and range queries. *IEEE Trans. Software Eng.*, 14(10):1381–1393, October 1988.
2. Faloutsos C. and Rong Y. Dot: a spatial access method using fractals. In Proc. 7th Int. Conf. on Data Engineering, 1991, pp. 152–159.
3. Gibson G., Nagle D., Amiri K., Butler J., Chang F.W., Gobioff H., Hardin C., Riedel E., Rochberg D., and Zelenka J. File server scaling with network-attached secure disks. In Proc. 1997 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comp. Syst., 1997, pp. 272–284.
4. Goldschlager L.M. Short algorithms for space-filling curves. *Software–Prac. Exper.*, 11(1):99–100, 1981.
5. Gray F. Pulse code communications. US Patent 2632058, 1953.
6. Jagadish H.V. Linear clustering of objects with multiple attributes. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 332–342.
7. Kamel I. and Faloutsos C. Hilbert r-tree: an improved r-tree using fractals. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 500–509.
8. Liao S., Lopez M.A., and Leutenegger S.T. High dimensional similarity search with space-filling curves. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 615–622.

9. Mokbel M.F., Aref W.G., El-Bassoumi K., and Kamel I. Scalable multimedia disk scheduling. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 498–509.
10. Mokbel M.F., Aref W.G., and Kamel I. Analysis of multi-dimensional space-filling curves. *GeoInformatica*, 7(3):179–209, September 2003.
11. Mokbel M.F. and Aref W.G. Irregularity in multi-dimensional space-filling curves with applications in multimedia databases. In Proc. Int. Conf. on Information and Knowledge Management, 2001, pp. 512–519.
12. Moon B., Jagadish H.V., Faloutsos C., and Salz J. Analysis of the clustering properties of hilbert space-filling curve. *IEEE Trans. Knowl. Data Eng.*, 13(1):124–141, 2001.
13. Orenstein J.A. Spatial query processing in an object-oriented database system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 326–336.
14. Peano G. Sur une courbe qui remplit toute une aire plane. *Math. Ann.*, 36:157–160, 1890.
15. Witten I.H. and Wyvill B. On the generation and use of space-filling curves. *Software–Prac. Exper.*, 3:519–525, 1983.

Space-Span (in Part)

▶ Context

Spamdexing

▶ Web Spam Detection

Span

▶ Time Interval

Sparse Index

MIRELLA M. MORO¹, VASSILIS J. TSOTRAS²

¹Federal University of Rio Grande do Sul,
Porto Alegre, Brazil

²University of California-Riverside, Riverside,
CA, USA

Synonyms

Non-dense Index

Definition

Consider a tree-based index on some numeric attribute A of a relation R . If an index record (of the form $\langle \text{search-key}, \text{pointer} \rangle$) is created for *some* of the values that appear in attribute A , then this index is *sparse*.

Key Points

Tree-based indices are built on numeric attributes and maintain an order among the indexed search-key values. Hence, they provide efficient access to the records of a relation by attribute value. Consider for example an index built on attribute A of relation R . The leaf pages of the index contain *index-records* of the form $\langle \text{search-key}, \text{pointer} \rangle$, where *search-key* corresponds to a value from the indexed attribute A and *pointer* points to the respective record in the indexed relation R with that attribute value. If not all distinct values that appear in $R.A$ also appear in index records, this index is *sparse*, otherwise it is called *dense*.

A sparse index needs a way to access even the relation records with values that do not directly appear in the index. Hence it is required that the indexed relation is *ordered* according to the values of the indexed attribute A ; in this way the relation order can be used to access values not directly indexed by the sparse index.

Tree-indices are further categorized by whether their search-key ordering is the same with the relation file's physical order (if any). If the search-key of a tree-based index is the same as the ordering attribute of a (ordered) file then the index is called *primary*. An index built on any non-ordering attribute of a file is called *secondary*. Hence a primary index is also sparse while a secondary index should also be dense.

A dense index is typically larger than a sparse index (since all search-key values are indexed) and thus requires more space. It also needs to be updated for every relation update that involves the attribute value being indexed.

Cross-references

- ▶ Access Methods
- ▶ B+-Tree
- ▶ Index Sequential Access Method (ISAM)
- ▶ ISAM
- ▶ Primary Index

Recommended Reading

1. Elmasri and Ramez Navathe. Shamkant B. Fundamentals of Database Systems (5th edn.). Addison-Wesley, Reading, MA, 2007.

2. Manolopoulos, Yannis Theodoridis, and Yannis Tsotras. Vassilis J. Advanced Database Indexing. Kluwer, Dordrecht, 1999.
3. Silberschatz, Avi Korth, and Henry F. S. Sudarshan Database System Concepts (5th edn.). McGraw-Hill, NY, 2006.

Spatial Access Methods

- ▶ Spatial Indexing Techniques

Spatial Analysis

- ▶ Spatial Data Analysis

Spatial and Spatio-Temporal Data Models and Languages

MARKUS SCHNEIDER

University of Florida, Gainesville, FL, USA

Definition

A *data model* provides a formalism consisting of a notation for describing data of interest and of a set of operations for manipulating these data. It abstracts from reality and provides a generalized view of data representing a specific and bounded scope of the real world. In the context of databases, a data model describes the organization, that is, the structure, of a database. In the context of complex objects like video, genomic, and multimedia objects, a data model describes a type system consisting of data types, operations, and predicates. Spatial and spatio-temporal data models are of this second kind. A *spatial data model* is a data model defining the properties of and operations on static objects in space. These objects are described by *spatial data types* like *point* (for example, representing the locations of cities in the US), *line* (for example, describing the ramifications of the Nile Delta), and *region* (for example, depicting school districts). Operations on spatial data types include, for instance, the geometric *intersection*, *union*, and *difference* of spatial objects, the computation of the *length* of a line or the *area* of a region, the test whether two spatial objects *overlap* or *meet*, and whether one object is *north* or *southeast* of another object. A *spatio-temporal data model* is a data model representing the temporal

evolution of spatial objects over time. These evolutions can be discrete, that is, they happen from time to time (for example, the change of the boundary of a land parcel) or continuous, that is, they happen permanently and smoothly (for example, the devastating trajectory of a hurricane). In the continuous case, one speaks about *moving objects* and represents them by *spatio-temporal data types* like *moving point* (for example, recording the route of a cell phone user), *moving line* (for example, representing the boundary of a tsunami), and *moving region* (for example, describing the motion of an air polluted cloud). Operations on spatio-temporal data types comprise, for instance, the spatio-temporal *intersection*, *union*, and *difference* of moving objects, the computation of the *trajectory* of a moving point as a line object, the determination of the *location* of a moving object at a particular time, the calculation of a moving object during a given set of intervals, and the test whether a moving point *enters* or *crosses* a moving region. *Spatial and spatio-temporal query languages* enable the user to query databases enhanced by these concepts.

Historical Background

The interest to store geometric data in databases began in the late 1970s. Due to the increasing success of relational databases, the first approach has been to decompose a spatial object recursively into its constituent parts until they can be stored in tables. For example, this approach decomposes a polygon into its set of segments. Each segment is decomposed into a pair of points. A point is decomposed into a pair of two float numbers. Float numbers are a DBMS data type and can be stored in a table. This approach has revealed a number of fundamental drawbacks. Since all lines and polygons are decomposed into their constituent parts scattered as tuples over a relation, a spatial object is not treated as an entity or unit but only *corresponds* to a collection of tuples. Since this approach is based on standard domains and has no concept of spatial

data types, it cannot provide and support any meaningful geometric operations. A more detailed discussion can be found in [7].

Classical research on time-varying geometric data has focused on *discrete* changes of spatial objects over time. For example, cadastral applications deal with the management of land parcels whose boundaries can change from time to time due to specific legal actions such as splitting, merging, or land consolidation. Political boundaries can suddenly disappear, as the reunification of West and East Germany shows. Different approaches have been proposed to model these discrete changes. One of them is to enhance *temporal databases* [12] with spatial data types. Each discrete change leads to a new stored snapshot with a modified spatial object in the temporal database. Another approach [15] keeps a single version of each spatial object only but annotates each of its components (for instance, a point or a segment) with a temporal element indicating the period of validity or existence of this component. Hence, discrete changes of a spatial object are registered within the object.

Foundations

Spatial data types form the basis of a large number of data models and query languages for spatial data. They are extensively leveraged by *spatial databases* [9] and embedded as attribute data types into their data models, that is, in the same way as standard data types such as *integer*, *real*, and *string*. The geometric types are designed as *abstract data types*, that is, the internal structure of a spatial object is hidden from the user, and its features can only be retrieved by (abstract) operations on this object. In this manner, they provide a high-level view of geometric data. One can distinguish the older generation of *simple* spatial data types and the newer generation of *complex* spatial data types, depending on the spatial complexity the types are able to model. In the two-dimensional space, simple spatial data types only provide simple object



Spatial and Spatio-Temporal Data Models and Languages. Figure 1. Examples of a simple point object (a), a simple line object (b), a simple region object (c), a complex point object (d), a complex line object (e), and a complex region object (f).

structures like single points, continuous lines, and simple regions (Fig. 1a–c). However, from an application perspective, simple spatial data types have turned out to be inadequate abstractions for spatial applications since they are insufficient to cope with the variety and complexity of geographic reality. From a formal perspective, they are not closed under the geometric set operations *intersection*, *union*, and *difference*. This means that these operations applied to two simple spatial objects can produce a spatial object that is *not* simple. Complex spatial data types solve these problems. They provide universal and versatile spatial objects and are closed under geometric set operations. They allow objects with multiple components, region components that may have holes, and line components that may model ramified, connected geometric networks (Fig. 1d–f).

As an example, in a relational setting, states, cities, and rivers are represented in the following relations:

```
states(sname: string, area: region)
cities(name: string, population: integer,
location: point)
rivers(name: string, route: line)
```

Queries can then be formulated by employing operations and predicates on spatial attribute values within an extended standard database query language such as SQL, leading to *Spatial SQL* [1]. Assume that the following operations and predicates are available:

<i>area</i> :	<i>region</i>	→ <i>real</i>
<i>inside</i> :	<i>point</i> × <i>region</i>	→ <i>bool</i>
<i>intersection</i> :	<i>line</i> × <i>region</i>	→ <i>line</i>
<i>length</i> :	<i>line</i>	→ <i>real</i>
<i>meet</i> :	<i>region</i> × <i>region</i>	→ <i>bool</i>

The predicates *inside* and *meet* represent *topological relationships* [8] that characterize the relative position between spatial objects. The operation *length* is a numerical function computing the length of a *line* object. The operation *intersection* computes the part of a *line* object intersecting a *region* object.

One can now pose queries: What is the total population of the cities in France?

```
select sum(c.pop) as total
from   cities as c, states as s
```

```
where c.location inside s.area and
s.name = 'France'
```

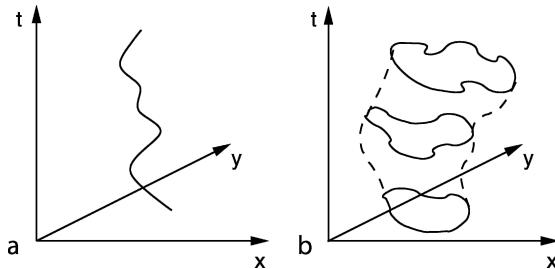
Compute the part of the river Rhine that is located within Germany and determine its length.

```
select intersection(r.route, s.area) as rhine,
       length(intersection(r.route,
                           s.area)) as len
  from rivers as r, states as s
 where r.name = 'Rhine' and s.name = 'Germany'
```

Make a list that shows for each state the number of its neighbor states, and their total area.

```
select s.name, count(*),
       sum(area(t.area))
  from states as s, states as t
 where s.area meet t.area
 group by s.name
```

Spatio-temporal data types enable the user to describe the dynamic behavior of spatial objects over time. The dynamic behavior refers to the continuous change of the locations of spatial objects over time. That is, the spatial objects move, and they are therefore called *moving objects*. They are stored in special spatio-temporal databases called *moving objects databases* [6]. In the same way as spatial data types, spatio-temporal data types are also designed as abstract data types and embedded as attribute types into a DBMS data model. Spatio-temporal data types are available for moving points (type *mpoint* for short), moving lines (*mline*), and moving regions (*mregion*). In case of moving regions, one can also represent the change of their extent and shape over time. Conceptually, a moving point is a function $f: \text{time} \rightarrow \text{point}$, a moving line is a function $f: \text{time} \rightarrow \text{line}$, and a moving region is a function $f: \text{time} \rightarrow \text{region}$. For example, for a moving region this means that at each time instant an object of type *region* has to be returned. Geometrically, moving objects correspond to the three-dimensional shapes. In case of a moving point it is a three-dimensional line (Fig. 2a) and in case of a moving region it is a volume (Fig. 2b). One can distinguish moving objects databases that model and query the history of movement for spatio-temporal analysis [2,5] and moving objects databases that model, predict, and query current and future movement [10,11]. In the latter case, location updates require a balancing of update costs and



Spatial and Spatio-Temporal Data Models and Languages. Figure 2. Examples of a moving point object (a) and a moving region object (b).

imprecision [14] and introduce the feature of uncertainty [13].

As an example, consider relations describing the movements of airplanes or storms:

```
flight(id: string, from: string,
      to: string, route: mpoint)
weather(id: string, kind:
      string, area: mregion)
```

One can pose queries by employing operations and predicates on spatio-temporal attribute values within an extended standard database query language such as SQL, leading to *Spatio-Temporal Query Language (STQL)* [3]. Assume that the following operations and predicates are available:

<i>deftime</i> :	<i>mpoint</i>	\rightarrow <i>periods</i>
<i>Disjoint</i> :	<i>mpoint</i> \times <i>mregion</i>	\rightarrow <i>bool</i>
<i>distance</i> :	<i>mpoint</i> \times <i>mpoint</i>	\rightarrow <i>mreal</i>
<i>Inside</i> :	<i>mpoint</i> \times <i>mregion</i>	\rightarrow <i>bool</i>
<i>intersection</i> :	<i>mpoint</i> \times <i>mregion</i>	\rightarrow <i>mpoint</i>
<i>meet</i> :	<i>point</i> \times <i>region</i>	\rightarrow <i>bool</i>
<i>min</i> :	<i>mreal</i>	\rightarrow <i>real</i>
<i>trajectory</i> :	<i>mpoint</i>	\rightarrow <i>line</i>

The function *deftime* returns the set of time intervals when a moving point is defined. The *spatio-temporal predicate* [3,4,6] *Disjoint* checks whether a moving point and a moving region are disjoint for some period. The function *distance* computes the distance between two moving points and is a real-valued function of time, captured here in a data type *mreal* for *moving reals*. The spatio-temporal predicate *Inside* tests whether a moving point is located inside a moving region for some period. The operation

intersection returns the part of a moving point whenever it lies inside a moving region, which is a moving point again. The topological predicate *meet* checks whether a point object is located on the boundary of a region object. The function *min* yields the minimal value assumed over time by a moving real.

One can now pose queries: Find all flights from Frankfurt that are longer than 5,000 kms.

```
select id
  from flight
 where from = 'FRA' and length
  (trajectory(route)) > 5000
```

Retrieve any pairs of airplanes, which, during their flight, came closer to each other than 500 m.

```
select f.id, g.id
  from flight as f, flight as g
 where f.id <> g.id and min(distance
  (f.route, g.route)) < 0.5
```

At what time was flight TB691 within a snowstorm with id RS316?

```
select      deftime(intersection(f.route,
                                     w.area))
  from      flight as f, weather as w
 where      f.id = 'TB691' and w.id = 'RS316'
```

Which are the planes that ran into a hurricane and had to traverse it?

```
select f.id, w.id
  from flight as f, weather as w
 where w.kind = 'hurricane' and
  f.route Disjoint >> meet >>
  Inside >> meet >> Disjoint w.area
```

The term *Disjoint* \gg *meet* \gg *Inside* \gg *meet* \gg *Disjoint* is a spatio-temporal predicate that is composed of a *temporal sequence* of the basic spatio-temporal predicates *Disjoint* and *Inside* as well as the topological predicate *meet*. The *temporal composition operator* is indicated by the symbol \gg . The query above searches for a spatio-temporal pattern in which a plane is disjoint from a hurricane for some period, then meets the boundary of the hurricane at a time instant, is inside the hurricane for some period, meets the boundary of the hurricane again at a time instant, and is disjoint again from the hurricane for some period. The alternating sequence of topological predicates that hold for

some period or for some time instant is characteristic for composite spatio-temporal predicates.

Key Applications

Spatial data models containing spatial data types, operations, and predicates are a universal and general concept for representing geometric information in all kinds of spatial applications. They have found broad acceptance in spatial extension packages of commercially and publicly available database systems as well as in geographic information systems. Further, all applications in the geosciences (for example, geography, hydrology, soil sciences) as well as many applications in government and administration (for example, cadastral applications, urban planning) already benefit from them. Independent studies have shown that about 80% of all data have spatial features (like geometric attributes) or a spatial reference (like an address). Thus, it is not surprising that independent international studies have predicted that geoinformation technology will belong to the most important and promising technologies in the future, besides biotechnology and nanotechnology.

The usage of moving objects in databases and especially in geographic information systems is still in its infancy since it is a relatively new technology. But increasingly, applications like location management, GPS-equipped PDAs, phones, and vehicles, navigation systems, RFID-tag tracking, sensor networks, hurricane research, and national security show interest in moving objects databases.

Cross-references

- ▶ [Spatial Data Types](#)
- ▶ [Spatio-temporal Trajectories](#)
- ▶ [Temporal Database](#)

Recommended Reading

1. Egenhofer M.J. Spatial SQL: a query and presentation language. *IEEE Trans. Knowl. and Data Eng.*, 6(1):86–94, 1994.
2. Erwig M., Güting R.H., Schneider M., and Vaziriannis M. Spatio-temporal data types: an approach to modeling and querying moving objects in databases, *Geoinformatica*, 3(3): 265–291, 1999.
3. Erwig M. and Schneider M. Developments in spatio-temporal query languages. In Proc. IEEE International Workshop on Spatio-Temporal Data Models and Languages, 1999, pp. 441–449.
4. Erwig M. and Schneider M. Spatio-Temporal Predicates. *IEEE Trans. Knowl. and Data Eng.*, 14(4):1–42, 2002.

5. Güting R.H., Böhlen M.H., Erwig M., Jensen C.S., Lorentzos N.A., Schneider M., and Vaziriannis M. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, 2000.
6. Güting R.H. and Schneider M. *Moving Objects Databases*. Morgan Kaufmann, San Francisco, CA, USA, 2005.
7. Schneider M. *Spatial Data Types for Database Systems – Finite Resolution Geometry for Geographic Information Systems*. LNCS 1288. Springer, Berlin Heidelberg New York, 1997.
8. Schneider M. and Behr T. Topological relationships between complex spatial objects. *ACM Trans. Database Syst.*, 31(1): 39–81, 2006.
9. Shekar S. and Chawla S. *Spatial Databases: A Tour*. Prentice-Hall, Englewood Cliffs, NJ, USA, 2003.
10. Sistla A.P., Wolfson O., Chamberlain S., and Dao S. Modeling and Querying Moving Objects. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 422–432.
11. Sistla A.P., Wolfson O., Chamberlain S., and Dao S. Querying the uncertain position of moving objects. In: *Temporal Databases: Research and Practice*. O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS 1399. Springer, Berlin Hiedelberg New York, 1998, pp. 310–337.
12. Tansel A.U., Clifford J., Gadia S., Jajodia S., Segev A., and Snodgrass R.T. (eds.). *Temporal Databases: Theory, Design, and Implementation*. Benjamin/Cummings, 1993.
13. Trajcevski G., Wolfson O., Hinrichs K., and Chamberlain S. Managing uncertainty in moving objects databases. *ACM Trans. Database Syst.*, 29(3):463–507, 2004.
14. Wolfson O., Chamberlain S., Dao S., Jiang L., and Mendez G. Cost and Imprecision in Modeling the Position of Moving Objects. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 588–596.
15. Worboys M.F. A unified model for spatial and temporal information. *Comput. J.*, 37(1):25–34, 1994.

Spatial Anonymity

PANOS KALNIS, GABRIEL GHINITA

National University of Singapore, Singapore,
Singapore

Synonyms

[Spatial k-anonymity](#); [Privacy-preserving spatial queries](#); [Anonymity in location-based services](#)

Definition

Let U be a user who is asking via a mobile device (e.g., phone, PDA) a query relevant to his current location, such as “find the nearest betting office.” This query can be answered by a Location Based Service (LBS) in a public web server (e.g., Google Maps, MapQuest),

which is not trustworthy. Since the query may be sensitive, U uses encryption and a pseudonym, in order to protect his privacy. However, the query still contains the exact location, which may reveal the identity of U . For example, if U asks the query within his residence, an attacker may use public information (e.g., white pages) to associate the location with U . Spatial k -Anonymity (SKA) solves this problem by ensuring that an attacker cannot identify U as the querying user with probability larger than $1/k$, where k is a user-defined anonymity requirement. To achieve this, a centralized or distributed anonymization service replaces the exact location of U with an area (called Anonymizing Spatial Region or ASR). The ASR encloses U and at least $k - 1$ additional users. The LBS receives the ASR and retrieves the query results for *any* point inside the ASR. Those results are forwarded to the anonymization service, which removes the false hits and returns the actual answer to U .

Historical Background

The embedding of positioning capabilities (e.g., GPS) in mobile devices has triggered several exciting applications. At the same time, it has raised serious concerns [1] about the risks of revealing sensitive information in location based services (LBS). An untrustworthy LBS may use public knowledge to relate a set of query coordinates to a specific user, even if the user-id is removed. In practice, users are reluctant to access a service that may disclose their political/religious affiliations or alternative lifestyles. Furthermore, users might be hesitant to ask innocuous queries such as “find the restaurants in my vicinity” since, once their identity is revealed, they may face unsolicited advertisements. Spatial k -Anonymity (SKA) aims at solving this problem.

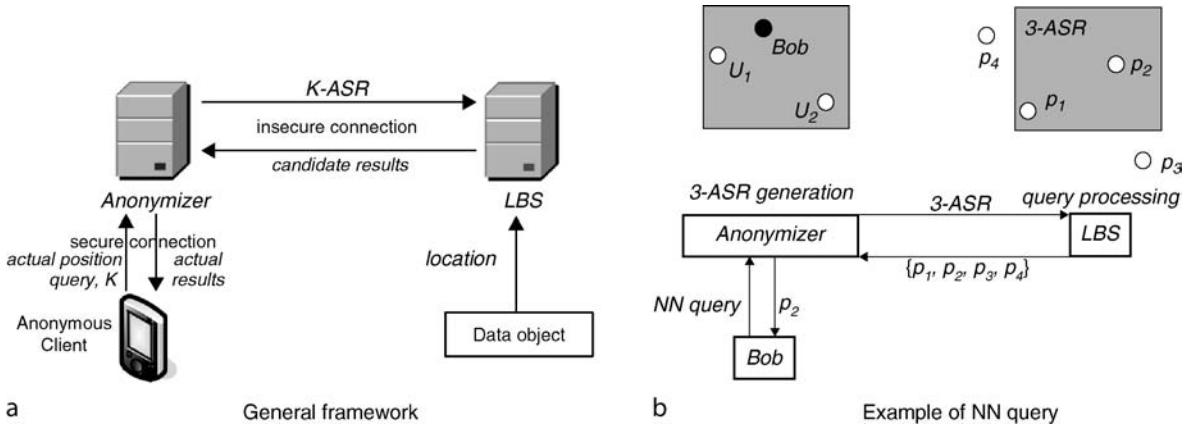
k -Anonymity [15] has been used in relational databases for publishing census, medical and voting registration data (often called microdata). A relation satisfies k -anonymity if every tuple is indistinguishable from at least $k - 1$ other tuples with respect to a set of quasi-identifier (QI) attributes. QIs are attributes (e.g., date of birth, gender, zip code) that can be linked to publicly available data to identify individuals. In the context of location based services, the k -Anonymity concept translates as follows: given a query, an attack based on the query location must not be able to identify the query source with probability larger than $1/k$.

A straightforward method is to pick $k - 1$ random users and forward k independent queries (including the real one) to the LBS. This method achieves SKA because the query could originate from any client with equal probability $1/k$. However, depending on the value of k , a potentially large number of locations are transmitted and processed by the LBS. Also, the exact locations of k users are revealed, which is undesirable in many applications.

Most of the existing work adopts the framework of Fig. 1a, which assumes a trusted server, called *anonymizer*. Users access the anonymizer through a secure connection and periodically report their position. A querying user U sends his location-based query to the anonymizer, which removes the user-id and transforms the location of U through a technique called cloaking. Cloaking hides the actual location by an *anonymizing spatial region* (k -ASR or ASR), which is an area that encloses U , as well as at least $k - 1$ other users. The anonymizer then sends the ASR to the LBS, which returns a set of *candidate* results that satisfy the query condition for any possible point inside the ASR.

Figure 1b presents an example, where Bob asks for the nearest betting office. Bob forwards his request to the anonymizer, together with his anonymity requirement k . Assuming that $k = 3$, the anonymizer generates a 3-ASR (shaded rectangle) that contains Bob and two other users U_1, U_2 (the anonymizer knows the exact locations of all users). Then, it sends this 3-ASR to the LBS, which finds all betting offices that can be the nearest-neighbor (NN) of any point in the 3-ASR (the LBS does not know where Bob is). This candidate set (i.e., $\{p_1, p_2, p_3, p_4\}$) is returned to the anonymizer, which filters the false hits and forwards the actual NN (in this case p_2) to Bob. Even if an attacker knows the location of Bob and the other users, she can only ascertain that the query originated from Bob with probability $1/3$.

The privacy of user locations has also been studied in the context of related problems. *Probabilistic Cloaking* [2] does not apply the concept of SKA; instead, the ASR is a closed region around the query point, which is independent of the number of users inside. Given an ASR, the LBS returns the probability of each candidate result satisfying the query, based on its location with respect to the ASR. Kamat et al. [10] propose a model for sensor networks and examine the privacy characteristics of different sensor routing protocols. Hoh and Gruteser [8] describe techniques for hiding the



Spatial Anonymity. Figure 1. Framework and example for spatial k -anonymity.

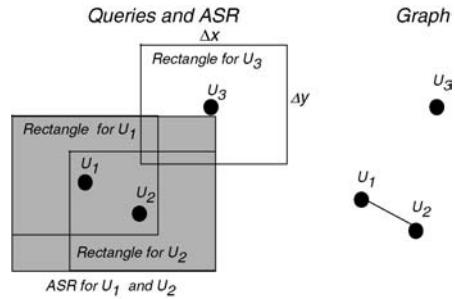
trajectory of users in applications that continuously collect location samples.

Foundations

In order to solve the SKA problem, the following assumptions about the capabilities of the attacker apply:

1. The attacker intercepts the ASR, which implies that either the LBS is not trustworthy, or the communication channel between the anonymizer and the LBS is not secure.
2. The attacker knows the cloaking algorithm used by the anonymizer. This is common in the security literature where algorithms are typically public.
3. The attacker can obtain the current locations of all users. This assumption is motivated by the fact that users may often issue queries from locations (e.g., home, office), which may be identified through physical observation, triangulation, telephone catalogs etc. However, it is difficult to model the exact amount of knowledge an attacker can gain. Therefore, the third assumption dictates that the anonymization method should be provably secure under the worst-case scenario.
4. The attacker uses current data, but not historical information about movement and behavior patterns of particular clients (e.g., a user often asking a particular query at a certain location or time). Therefore, SKA is defined only for *snapshot*, but not for continuous queries.

Given these assumptions, a spatial cloaking algorithm is said to be *secure*, if for any U and any k the probability of identifying U as the querying user is at most $1/k$.



Spatial Anonymity. Figure 2. Example of *Clique Cloak*.

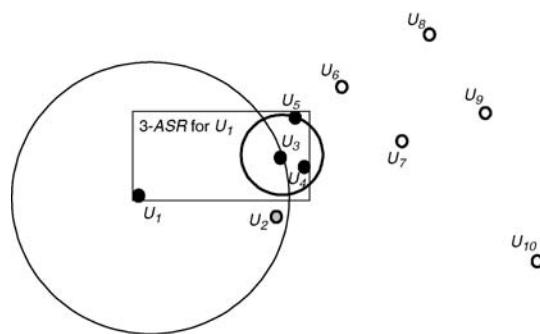
In addition to being secure, spatial cloaking should be efficient and effective. *Efficiency* means that the CPU and I/O cost of generating the ASR (at the anonymizer) should be minimized for better scalability and faster service. *Effectiveness* refers to the area of the ASR, which should also be minimized. Specifically, a large ASR incurs high processing overhead (at the LBS) and network cost (for transferring numerous candidate results from the LBS to the anonymizer). The rest of this article discusses several representative algorithms for the SKA problem.

In *Clique Cloak* [5], each query defines an axis-parallel rectangle whose centroid lies at the user location and whose extents are Δx , Δy . Figure 2 illustrates the rectangles of three queries located at U_1 , U_2 , U_3 , assuming that they all have the same Δx and Δy . The anonymizer generates a graph where a vertex represents a query: two queries are connected if the corresponding users fall in the rectangles of each other. Then, the graph is searched for cliques of k vertices and the minimum

bounding rectangle (*MBR*) of the corresponding user areas, forms the ASR sent to the LBS. Continuing the example of Fig. 2, if $k = 2$, U_1 and U_2 form a 2-clique and the MBR of their respective rectangles is generated so that both queries are processed together. On the other hand, U_3 cannot be processed immediately, but it has to wait until a new query (generating a 2-clique with U_3) arrives. *Clique Cloak* allows users to specify a temporal interval Δt such that, if a clique cannot be found within Δt , the query is rejected. Therefore, *Clique Cloak* may affect the quality of service, as some queries may be delayed or completely rejected. The algorithms discussed next do not suffer from this drawback.

Simply generating an ASR that includes k clients is not sufficient for SKA. Consider for instance an algorithm, called *Center Cloak* in the sequel, that given a query from U , finds his $k - 1$ closest users and sets the ASR as the MBR that encloses them. In fact, a similar technique is proposed in [4] for anonymization in peer-to-peer systems (i.e., the ASR contains the query issuing peer and its $k - 1$ nearest nodes). However, by construction, the querying user U is often closest to the ASR center. Thus, a simple “center-of-ASR” attack would correctly guess U with probability that far exceeds $1/k$, especially for large k values.

Nearest Neighbor Cloak (*NN-Cloak*) [9] is a randomized variant of *Center Cloak*, which is not vulnerable to the *center-of-ASR attack*. Given a query from U , *NN-Cloak* first determines the set S_0 containing U and his $k - 1$ nearest users. Then, it selects a random user U_i from S_0 and computes the set S_1 , which includes U_i and his $k - 1$ nearest neighbors (NN). Finally, *NN-Cloak* obtains $S_2 = S_1 \cup U$. This step is essential, since U

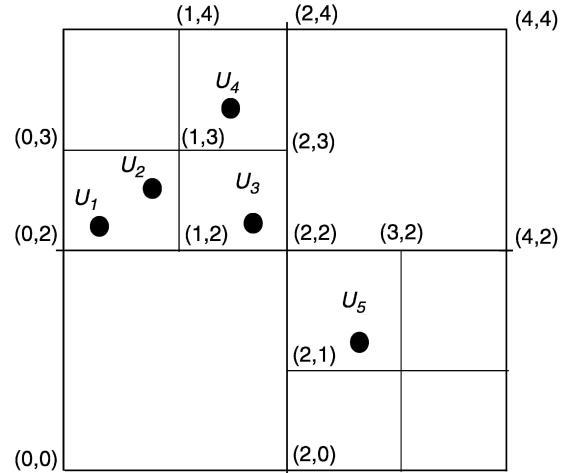


Spatial Anonymity. Figure 3. Example of *NN-Cloak*.

is not necessarily among the NNs of U_i . The ASR is the MBR enclosing all users in S_2 . Figure 3 shows an example of *NN-Cloak*, where U_1 issues a query with $k = 3$. The two NNs of U_1 are U_2, U_3 , and $S_0 = \{U_1, U_2, U_3\}$. *NN-Cloak* randomly chooses U_3 and issues a 2-NN query, forming $S_1 = \{U_3, U_4, U_5\}$. The 3-ASR is the MBR enclosing $S_2 = \{U_1, U_3, U_4, U_5\}$. It is not vulnerable to the *center-of-ASR attack* since the probability of U being near the center of the ASR is at most $1/k$ (due to the random choice).

In another approach called *Casper* [12], the anonymizer maintains the locations of the clients using a pyramid data structure, similar to a Quad-tree [14], where the minimum cell size corresponds to the anonymity resolution. Once the anonymizer receives a query from U , it uses a hash table on the user-id pointing to the lowest-level cell c where U lies. If c contains enough users (i.e., $|c| \geq k$), it becomes the ASR. Otherwise, the horizontal c_h and vertical c_v neighbors of c are retrieved. If the union of c with c_h or c_v contains at least k users, the corresponding union becomes the ASR. Else, *Casper* retrieves the parent of c and repeats this process recursively. Figure 4 shows an example. Cells are denoted by the coordinates of their lower-left and upper-right points. Assume a query q with $k = 2$. If q is issued by U_1 or U_2 , the ASR is cell $\langle(0,2), (1,3)\rangle$. If q is issued by U_3 or U_4 , the ASR is the union of cells $\langle(1,2), (2,3)\rangle \cup \langle(1,3), (2,4)\rangle$. Finally, if q is issued by U_5 , the ASR is the entire data space.

Interval Cloak [7] is similar to *Casper* in terms of both the data structure used by the anonymizer (i.e., a



Spatial Anonymity. Figure 4. Example of *Casper*.

Quad-tree) and the cloaking algorithm. The main difference is that *Interval Cloak* does not consider neighboring cells at the same level when determining the ASR, but ascends directly to the ancestor level. For instance, a query with $k = 2$ issued by U_3 or U_4 , would generate the ASR $\langle(0,2),(2,4)\rangle$ (instead of $\langle(1,2),(2,4)\rangle$ for *Casper*).

An important observation is that *Casper* and *Interval Cloak* are secure only for uniform data [8]. In the example of Fig. 4, although U_1 to U_4 are in the 2-ASR of U_5 , U_5 is not in the 2-ASR of any of those users. Consequently, an attacker that detects an ASR covering the entire space can infer with high probability that it originates from U_5 (by assumption 3, in the worst case the attacker may know the locations of all users). *NN-Cloak* also faces similar problems. Kalnis et al. [9] identified reciprocity as a sufficient property for secure algorithms. Formally:

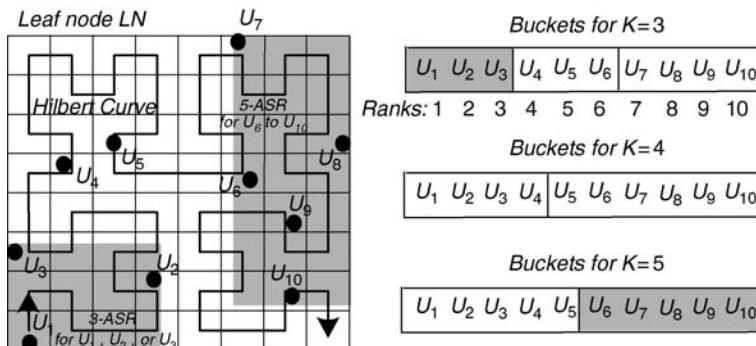
Definition [Reciprocity]. Let U be any user who is issuing a query with anonymity degree k , a set of users AS called anonymizing set, and a region ASR which encloses the users in AS. AS satisfies the reciprocity property if (i) it contains U and at least $k - 1$ additional users, and (ii) every user in AS also generates the same anonymizing set AS for the given k .

The only existing reciprocal (i.e., secure) algorithm is *Hilbert Cloak* [9], which has also been implemented on a peer-to-peer system for distributed anonymization [6]. *Hilbert Cloak* sorts the users according to their Hilbert value. The Hilbert space filling curve [13] transforms the multi-dimensional coordinates of each user U into an 1-D value $H(U)$. Figure 5 illustrates the Hilbert curve for a 2-D space using a 8×8 space partitioning. A user U is assigned the value $H(U)$ of

the cell that covers him. If two users are near each other in the 2-D space, they are likely to be close in the 1-D transformation. Given a query with anonymization degree k , *Hilbert Cloak* assigns the first k users (in the Hilbert order) to the first bucket, the next k users to the second bucket and so on. Following this approach, each bucket contains exactly k users, except for the last one that may include up to $2k - 1$. Let $r(U)$ be the rank of U in the Hilbert order. The bucket b_U of U contains all clients whose ranks are in the range $[s, e]$, where $s = r(U) - (r(U) - 1) \bmod k$ and $e = s + k - 1$ (unless b_U is the last bucket). The example of Fig. 5 contains 10 users, whose user-ids are ordered according to their Hilbert value. Consider a query from U_7 with $k = 5$. The rank of U_7 is $r(U_7) = 7$. The bucket containing U_7 starts at $s = 7 - 6 \bmod 5 = 6$ and ends at $e = 10$, i.e., it contains all users from U_6 to U_{10} . Its ASR is the MBR (shaded rectangle at the upper-right corner) covering the corresponding users. Any query with $k = 5$ originating from these users will generate the same b_U and ASR, thus guaranteeing reciprocity. It must be noted that *Hilbert Cloak* constructs on-the-fly only b_U as the remaining buckets are irrelevant to the query. Figure 5 illustrates another ASR (shaded rectangle at the lower-left corner) for a query with $k = 3$ originating from one of U_1 to U_3 .

Key Applications

In recent years, positioning devices (e.g., GPS) have gained tremendous popularity. Navigation systems are already widespread in the automobile industry and, together with wireless communications, facilitate exciting new applications. General Motors OnStar system, for example, supports on-line rerouting to avoid traffic jams



Spatial Anonymity. Figure 5. Example of *Hilbert Cloak*.

and automatically alerts the authorities in case of an accident. More applications based on the user locations are expected to emerge with the arrival of the latest gadgets (e.g., iPAQ hw6515, Mio A701), which combine the functionality of a mobile phone, PDA and GPS receiver. For such applications to succeed, the privacy and confidentiality issues are of paramount importance.

Future Directions

Existing methods focus on snapshot queries. An interesting problem concerns continuous SKA [3]. In this setting, a client poses a long running query about its surroundings (e.g., “find the nearest gas station”), whose results are updated as the client moves. The cloaking algorithm should generate a continuously changing ASR in a way that does not reveal information about the user through inspection of the individual ASR snapshots. Another future direction involves the elimination of the anonymizer layer by employing private information retrieval techniques. Khoshgozaran and Shahabi [11] present an initial approach which employs 1-D transformation and encryption to conceal both the spatial data and the queries from the LBS.

Experimental Results

Compared to *Interval Cloak*, *Casper* is better both in terms of efficiency (i.e., ASR generation cost) and effectiveness (i.e., ASR size). *NN-Cloak* is worse than *Casper* in terms of efficiency, but it is considerably better in terms of effectiveness. The choice of cloaking algorithm depends on the application characteristics. If, for instance, the anonymizer charges clients according to their usage and the LBS is a public (i.e., free) service, efficiency is more important. On the other hand, if the LBS imposes limitations (e.g., on the number of results, processing time, etc) effectiveness becomes the decisive factor. Finally, *Hilbert Cloak* is very similar to *Casper* both in terms of efficiency and effectiveness. However, *Hilbert Cloak* is the only provably secure method.

Cross-references

- [Anonymity](#)
- [Location-Based Services](#)
- [Nearest Neighbor Query](#)
- [Privacy](#)

► [Quadtrees \(and Family\)](#)

► [Space-Filling Curve](#)

Recommended Reading

1. Beresford A.R. and Stajano F. Location privacy in pervasive computing. *IEEE Pervasive Comput.*, 2(1):46–55, 2003.
2. Cheng R., Zhang Y., Bertino E., and Prabhakar S. Preserving user location privacy in mobile data management infrastructures. In *Proceedings of Privacy Enhancing Technologies*, 2006, pp. 393–412.
3. Chow C.-Y. and Mokbel M.F. Enabling private continuous queries for revealed user locations. In *Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases*, 2007, pp. 258–275.
4. Chow C.-Y., Mokbel M.F., and Liu X. A peer-to-peer spatial cloaking algorithm for anonymous location-based services. In *Proc. 14th ACM Int. Symp. on Geographic Inf. Syst.*, 2006, pp. 171–178.
5. Gedik B. and Liu L. Location privacy in mobile systems: a personalized anonymization model. In *Proc. 23rd Int. Conf. on Distributed Computing Systems*, 2005, pp. 620–629.
6. Ghinita G., Kalnis P., and Skiatopoulos S. PRIVE: anonymous location-based queries in distributed mobile systems. In *Proc. 16th Int. World Wide Web Conference*, 2007, pp. 371–380.
7. Gruteser M. and Grunwald D. Anonymous usage of location-based services through spatial and temporal cloaking. In *Proc. 1st Int. Conf. Mobile Systems, Applications and Services*, 2003, pp. 31–42.
8. Hoh B. and Gruteser M. Protecting location privacy through path confusion. In *Proc. 1st Int. Conf. on Security and Privacy for Emerging Areas in Communication Networks*, 2005.
9. Kalnis P., Ghinita G., Mouratidis K., and Papadias D. Preventing location-based identity inference in anonymous spatial queries. *IEEE Trans. Knowledge and Data Eng.*, 19(12):1719–1733, 2007.
10. Kamat P., Zhang Y., Trappe W., and Ozturk C. Enhancing source-location privacy in sensor network routing. In *Proc. 23rd Int. Conf. on Distributed Computing Systems*, 2005, pp. 599–608.
11. Khoshgozaran A. and Shahabi C. Blind evaluation of nearest neighbor queries using space transformation to preserve location privacy. In *Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases*, 2007, pp. 239–257.
12. Mokbel M.F., Chow C. Y., and Aref W.G. The new Casper: query processing for location services without compromising privacy. In *Proc. 32nd Int. Conf. on Very Large Data Bases*, 2006, pp. 763–774.
13. Moon B., Jagadish H.V., and Faloutsos C. Analysis of the clustering properties of the hilbert space-filling curve. *IEEE Trans. Knowledge and Data Eng.*, 13(1):124–141, 2001.
14. Samet, H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, New York, 1990.
15. Sweeney L. k-Anonymity: a model for protecting privacy. *Int. J. Uncertain., Fuzziness Knowledge-based Syst.*, 10(5):557–570, 2002.

Spatial Autocorrelation

- ▶ Spatial Data Mining

Spatial Data

- ▶ Query Evaluation Techniques for Multidimensional Data

Spatial Data Analysis

MICHAEL F. GOODCHILD

University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Spatial analysis; Geographical data analysis; Geographical analysis

Definition

Methods of data analysis perform logical or mathematical manipulations on data in order to test hypotheses, expose anomalies or patterns, or create summaries or views that expose particular traits. Data often refer to specific locations in some space. To qualify as spatial, the locations must be known and must affect the outcome of the analysis. While many spaces might be relevant, including the space of the human brain or the space of the human genome, the history of spatial data analysis is dominated by location in geographic space, in other words location on or near the surface of the Earth. Thus, geographical and spatial are often essentially synonymous. More formally, spatial data analysis can be defined as a set of techniques devised for the manipulation of data whose outcomes are not invariant under relocation of the objects of interest in some space. The term *exploratory spatial data analysis* (ESDA) describes an important subset that emphasizes real-time interaction, the creation of multiple views of data, the search for patterns and anomalies, and the generation of new hypotheses as opposed to the formal testing of existing ideas. The term *spatial data mining* describes another important subset that emphasizes the analysis of very large volumes of spatial data.

Historical Background

Berry and Marble [2] made one of the earliest efforts to assemble a systematic review of methods of spatial data analysis, drawing on a literature that had accumulated for many decades. Their interest was sparked in large part by what later became known as the Quantitative Revolution in Geography, a paradigm shift that originated at the University of Washington in the late 1950s and spread rapidly as the original group of graduate students found faculty positions. Bunge [3] summarized the core concept: that the analysis of patterns of phenomena on the Earth's surface could lead to a set of formal theories about the behavior of human and natural systems, and that the discovery of such theories would put the discipline of geography on a sound scientific footing. Substantial progress was made in the 1960s, particularly in the study of patterns of settlement and economic activity, and in the study of such physical phenomena as meandering rivers and stream channel networks.

Beginning in the 1960s, the development of geographic information systems (GIS) provided a major impetus, by creating a simple structure in which methods of spatial data analysis could be implemented. By the 1980s, GIS had become a popular and rapidly growing software application, with a flourishing industry and tools to enable spatial data analysis, along with the necessary techniques for data acquisition, editing, and display. Today, GIS is often portrayed as an engine for spatial data analysis, and many new techniques have been added to what are now literally thousands of methods. GIS finds application in virtually all disciplines that deal with the surface and near-surface of the Earth, ranging from ecology and geology to sociology and political science [7]. It is extensively used in logistics, in planning and public decision making, in military and intelligence applications, and in the management of utility networks.

While the use of computers to perform spatial data analysis was already well established in the 1960s, ESDA emerged rather later, when the graphics and interactive capabilities of computers had advanced sufficiently. By the early 1990s, researchers were developing novel ways of linking multiple views using the windowing techniques that emerged at that time, and exploiting the high-resolution graphics that became available on standard personal computers. Today, interactive tools inspired by ESDA are widely available in GIS products, and more specialized software is

also available (see, for example, GeoDa, <http://geoda.uiuc.edu>).

Interest in spatial data mining has grown in the past decade, driven in part by the increasing availability of very large volumes of spatial data. For example, it is now routine to capture the location and time of use of credit and debit cards, and to apply sophisticated algorithms in an effort to detect fraudulent use. Heavy use of spatial data analysis is made by intelligence agencies, based on software that can examine telephone and email traffic and detect references to places.

Foundations

Several approaches have been devised for organizing the thousands of techniques that qualify as spatial data analysis. Perhaps the commonest, represented by several recent textbooks and by the organization of some GIS user interfaces, is based on a taxonomy of spatial data types. Very broadly, one can capture variation within a space using either raster or vector structures; a raster structure is created by dividing the space into discrete, regularly shaped elements and describing the contents of each, while vector structures describe each feature present in the space as either a point, line, area, or volume, with associated attributes.

Tomlin [9] and others have systematized the analysis of raster data in schemata described as map algebras, image algebras, or cartographic modeling, and several GIS have adopted these schemata in their user interfaces. In one such schema the analysis of raster data are described as either focal, local, zonal, or global: focal operations are performed independently on the contents of each cell; local operations are performed on a cell and its immediate neighborhood; zonal operations apply to contiguous patches of cells with identical descriptions; and global operations apply to the entire raster.

To date a similarly simple systematization of vector operations has not been achieved. Instead, several textbooks (e.g., [1,5]) organize methods of vector-based analysis according to the types of features being analyzed, focusing in turn on points, lines, areas, and volumes. For example, techniques for the analysis of sets of points might determine the degree of dispersion of the points; search for anomalous clusters; or find a shortest tour through the points. Some texts also provide descriptions of methods for the analysis of relationships or interactions between features. For example, retailers and traffic engineers commonly use

methods of spatial data analysis to predict the numbers of trips expected between home neighborhood areas and such destination points as shopping centers or places of work.

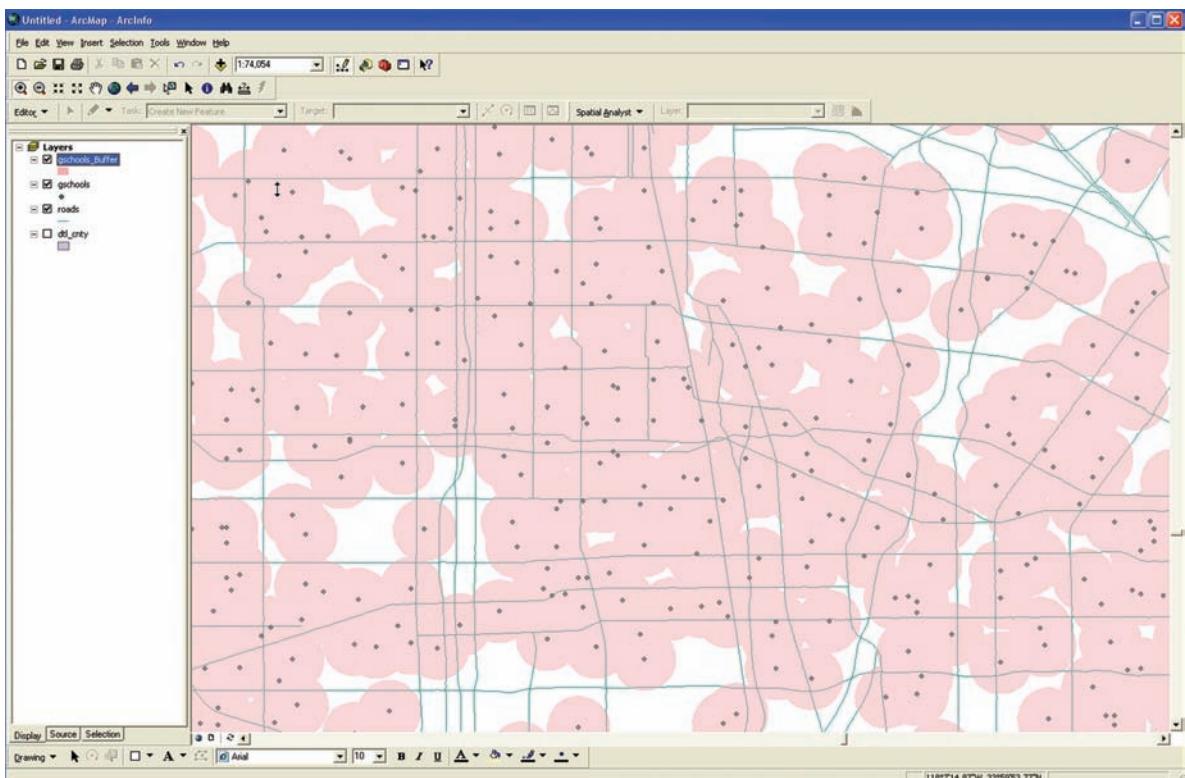
Longley et al. [7] use a different organizing scheme that is designed to be more strongly related to user motivation, and to overcome some of the ambiguities inherent in an emphasis on data type. Their scheme assigns techniques to six categories, ordered by increasing conceptual sophistication: query and reasoning, measurement, transformation, descriptive summary, optimization, and hypothesis testing.

Query and reasoning functions rely on the presentation of alternative views to the user. For example, a set of data on average income by US state might be presented as a map, as a table, as a histogram, and as a scatterplot in which average income is graphed against another variable such as percent with more than high-school education. The user gains insight by examining the alternative views, by querying specific values, or by selecting data items in one window and seeing them highlighted in the other windows.

Measurement functions represent one of the earliest motivations for GIS. Manual methods for obtaining measures of such properties as area, length, slope, or shape from maps are notoriously inaccurate, tedious, and time-consuming, whereas it is trivial to obtain them from digital representations. Nevertheless, digital representations are only approximations or generalizations of real phenomena, and many estimates exhibit representation-related biases.

Transformation functions obtain new objects, or new properties of those objects. They include many key GIS functions, including buffering (the geometric dilation of points, lines, areas, or volumes), overlay (the computation of intersections between objects), and interpolation (the use of data from sample locations to estimate values at locations where no samples were taken). Figure 1 shows an example of buffering, using half-mile circles around points representing the schools of part of Los Angeles. The example was motivated by proposals to ban registered sex offenders from living within a specified distance of a school.

Descriptive summaries include the widest range of spatial data analysis techniques. Standard univariate statistics such as the mean, median, mode, standard deviation, and variance have equivalents in multidimensional spaces. Figure 2 shows the two-dimensional equivalents of the mean and standard deviation

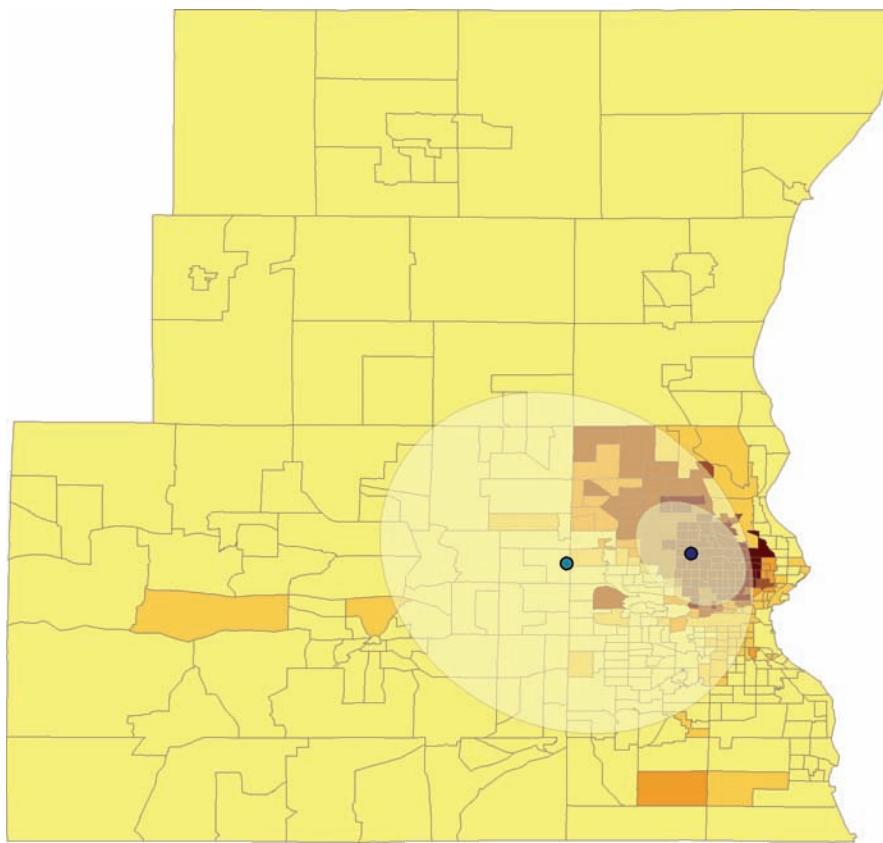


Spatial Data Analysis. Figure 1. The buffer operation. Half-mile buffers have been drawn around points representing the locations of schools in an area of central Los Angeles. Such buffers are often required by legislation; this example was motivated by a proposal to ban registered sex offenders from living within a prescribed distance of schools.

applied to the black and white populations of Milwaukee, using data by census tract. A suite of summary statistics have been devised for measuring *spatial dependence*, a key property of many spaces based on the observation that measurements of many properties taken close together tend to be more similar than measurements taken far apart. The fields of *spatial statistics* and *geostatistics* are both based on this property, and provide ways of addressing it explicitly. *Spatial heterogeneity*, or the tendency for the properties of spaces to vary widely from one area to another, is also the subject of many forms of descriptive summary. The rapidly evolving field of *local* or *place-based* summaries (e.g., [4]) addresses the spatial heterogeneity property directly, arguing that it is more important to determine how the results of spatial data analysis vary from one area to another than to attempt to extract single, global results. Another suite of summary statistics addresses the *fragmentation* of landscapes, with particularly strong applications in ecology.

Methods of *optimization* focus on the design of spatial pattern rather than on its analysis. They include methods for optimum location of points (e.g., retail stores, schools), lines (e.g., roads, pipelines), and areas (e.g., political voting districts), as well as on the design of optimum routes (e.g., for delivery vehicles or school buses).

Finally, methods of *hypothesis testing* address the process of inference, by which analysts reason from the analysis of a sample to conclusions about the larger world represented by the sample. Such methods are well known in statistical analysis, encompassing many well-known statistical tests, significance levels, and the formulation of null hypotheses. Unfortunately the application of such methods to spatial data is confounded by two major issues. First, it is rare for a sample of objects to be representative of any larger and well-defined set; instead, spatial data analysis is commonly applied to all of the objects that exist in a given study area. Second, it is also rare for objects distributed in space to be selected



Spatial Data Analysis. **Figure 2.** Two-dimensional equivalents of the mean and standard deviation. The larger ellipse shows the dispersion of the white population of Milwaukee around its centroid; the smaller ellipse shows the greater concentration of the city's black population. The map shows percent black, using 1990 data by census tract.

independently from any larger set; instead, the property of spatial dependence virtually ensures that nearby objects will have some degree of similarity.

Key Applications

As noted earlier, techniques of spatial data analysis can be applied to virtually all spaces, and all phenomena distributed within such spaces. Nevertheless, the vast majority of applications are found in geographic space, that is, the space defined by the surface and near-surface of the Earth, at spatial resolutions ranging from sub-meter to global.

Many important applications have derived from the need to understand the mechanisms of disease, and particularly its transmission within human populations. The work of Dr. John Snow on cholera [8] is often cited as the seminal example, but today methods of spatial data analysis are routinely used to scan data on such

diseases as cancer, searching for anomalous clusters and thus for potential causal mechanisms. Spatial data analysis has been central to the study of outbreaks of new diseases such as West Nile virus and SARS.

Spatial data analysis has also been central to the study of landscape change, and related phenomena of urban sprawl, deforestation, desertification, and habitat destruction. Such analyses are often based on snapshots of landscape obtained from Earth-orbiting satellites, and can form the basis for sophisticated models of landscape change that can be used to investigate the future effects of management alternatives.

Transportation applications are also particularly rich. Methods of spatial data analysis are routinely used to model traffic patterns, and to evaluate planning options, including new roads, mass transit, and congestion pricing. The possibility of real-time tracking of vehicles using GPS has recently given this field new impetus.

FUTURE DIRECTIONS

The insights that can be obtained from spatial data analysis are limited by its essentially cross-sectional nature – the need to draw inferences from snapshots obtained at one point in time. It is difficult, for example, to ascribe cause when no information is available about change through time. Thus there is great interest in the development of an improved suite of methods for *spatiotemporal* data analysis. In the past, the lack of suitable data has been a major impediment, but today vast new sources are becoming available as the result of developments in satellite remote sensing, GPS tracking, and Internet-based data sharing.

GIS owes much of its original stimulus to the paper map, which is of necessity flat. At global scales, analysis based on flattened or *projected* views of the Earth's surface can be misleading, and there is therefore strong interest in developing methods of spatial data analysis for the Earth's curved surface. This interest has been stimulated in part by the recent emergence of *virtual globes*, including Google Earth.

CROSS-REFERENCES

- ▶ [Geographic Information System](#)
- ▶ [Spatial Data Mining](#)
- ▶ [Spatial Data Types](#)
- ▶ [Spatial Operations and Map Operations](#)

RECOMMENDED READING

1. Bailey T.C. and Gatrell A.C. *Interactive Spatial Data Analysis*. Longman, 1995.
2. Berry B.J.L. and Marble D.F. *Spatial Analysis: A Reader in Statistical Geography*. Prentice Hall, Englewood Cliffs, NJ, 1968.
3. Bunge W. *Theoretical Geography*. University of Lund. Gleerup, Sweden, 1966.
4. Fotheringham A.S., Brunsdon C., and Charlton M. *Geographically Weighted Regression: The Analysis of Spatially Varying Relationships*. Wiley, 2002.
5. Haining R.P. *Spatial Data Analysis: Theory and Practice*. Cambridge University Press, UK, 2003.
6. Johnson S. *The Ghost Map: The Story of London's Most Terrifying Epidemic and How It Changed Science, Cities, and the Modern World*. Riverhead, 2006.
7. Longley P.A., Goodchild M.F., Maguire D.J., and Rhind D.W. *Geographic Information Systems and Science*. Wiley, New York, 2005.
8. O'Sullivan D., Unwin D.J. *Geographic Information Analysis*. Wiley, 2003.
9. Tomlin C.D. *Geographic Information Systems and Cartographic Modeling*. Prentice Hall, Englewood Cliffs, NJ, 1990.

Spatial Data Mining

SHASHI SHEKHAR, JAMES KANG, VIJAY GANDHI
University of Minnesota, Minneapolis, MN, USA

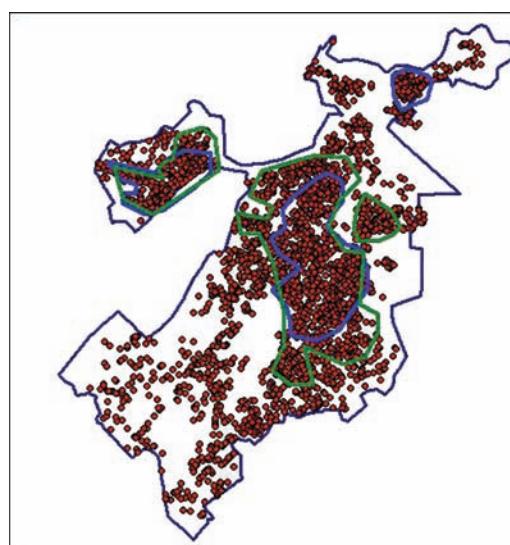
Synonyms

[Spatial data analysis](#); [Spatial statistics](#); [Co-locations](#); [Spatial outliers](#); [Hotspots](#); [Location prediction](#); [Spatial autocorrelation](#)

Definition

Spatial data mining is the process of discovering non-trivial, interesting, and useful patterns in large spatial datasets. The most common spatial pattern families are co-locations, spatial hotspots, spatial outliers, and location predictions.

[Figure 1](#) gives an example of a spatial hotspot pattern for burglary related crimes in the Boston, MA area. In this figure, each point depicts a burglary event in the year 1999. The dark blue and green shapes in the figure represent the discovered hotspots or the source of this type of crime. Notice that discovering these hotspots is a non-trivial process due to the irregular size and spatial shape of the pattern. In addition, not all incidents contribute to the hotspot. Discovery of these patterns is very useful and interesting to public safety professionals



Spatial Data Mining. Figure 1. Spatial hotspots of crimes in Boston, MA (best viewed in color) (courtesy: NIJ, Infotech).

as they plan police patrols and social interventions to reduce future crime incidents in the area.

Historical Background

Spatial data mining research began several decades ago when practitioners and researchers noticed that critical assumptions in classical data mining and statistics were violated by spatial datasets. First, whereas classical datasets often assume that data are discrete, spatial data were observed to reside in continuous space. For example, classical data mining and statistical methods may use market-basket datasets (e.g., history of Walmart's transactions), where each item-type in a transaction is discrete. However, "transactions" are not natural in continuous spatial datasets, and decomposing space across transactions leads to loss of information about neighbor relationships between items across transaction boundaries. In addition, spatial data often exhibits heterogeneity (i.e., no places on the Earth are identical), whereas classical data mining techniques often focus on spatially stationary global patterns (i.e., ignoring spatial variations across locations). Finally, one of the common assumptions in classical statistical analysis is that data samples are independently generated. However, this assumption is generally false when analyzing spatial data, because spatial data tends to be highly self-correlated. For example, people with similar characteristics, occupation and background tend to cluster together in the same neighborhoods. In spatial statistics [1] this tendency is called spatial auto-correlation. Ignoring spatial auto-correlation when analyzing data with spatial characteristics may produce hypotheses or models that are inaccurate or inconsistent with the data set. Thus, classical data mining algorithms often perform poorly when applied to spatial data sets. Better methods are needed to analyze spatial data to detect spatial patterns.

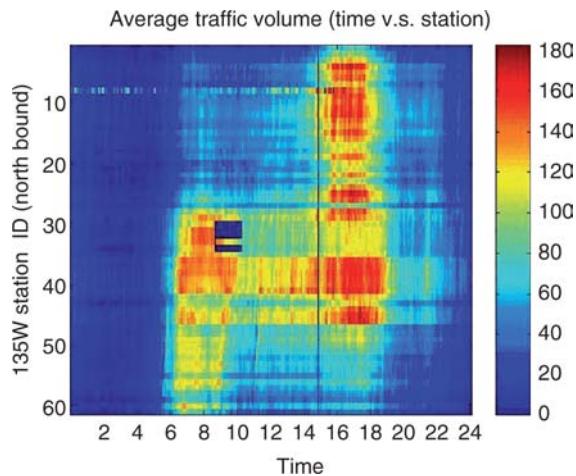
Foundations

The spatial data mining literature has focused on four main types of spatial patterns: (i) spatial outliers, which are spatial locations showing a significant difference from their neighbors; (ii) spatial co-locations, or subsets of event types that tend to be found more often together throughout space than other subsets of event types; (iii) location predictions, that is, information that is inferred about locations favored by an event type based on other explanatory spatial variables; and (iv) spatial hotspots, unusual spatial groupings of

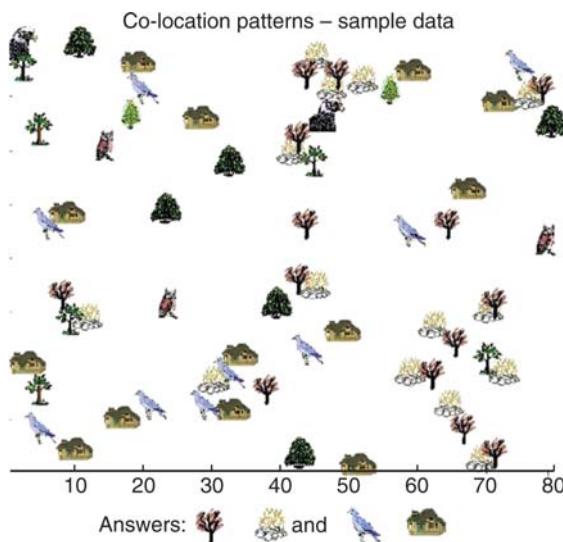
events. The remainder of this section presents a general overview of each of these pattern categories.

A *spatial outlier* is a spatially referenced object whose non-spatial attribute values differ significantly from those of other spatially referenced objects in its spatial neighborhood. Figure 2 gives an example of spatial outliers, detected in traffic measurements for sensors on highway I-35W (North bound) in the Minneapolis-St. Paul area, for a 24-h time period. Station 9 may be considered a spatial outlier as it exhibits inconsistent traffic flow compared with its neighboring stations. Once a spatial outlier is identified, one may proceed with diagnosis. For example, the sensor at Station 9 may be diagnosed as malfunctioning. Spatial attributes are used to characterize location, neighborhood, and distance. Non-spatial attribute dimensions are used to compare a spatially referenced object to its neighbors. Spatial statistics literature provides two kinds of bi-partite multidimensional tests, namely graphical tests and quantitative tests. Graphical tests, such as Variogram clouds and Moran scatter-plots, are based on the visualization of spatial data and highlights spatial outliers. Quantitative methods provide a precise test to distinguish spatial outliers from the remainder of data.

Spatial co-location pattern discovery finds frequently co-located subsets of spatial event types given a map of their locations. Figure 3 gives an example map with two examples of spatial co-locations. Readers are encouraged to determine for themselves the co-located pairs of spatial event types in Fig. 3. The answers provided



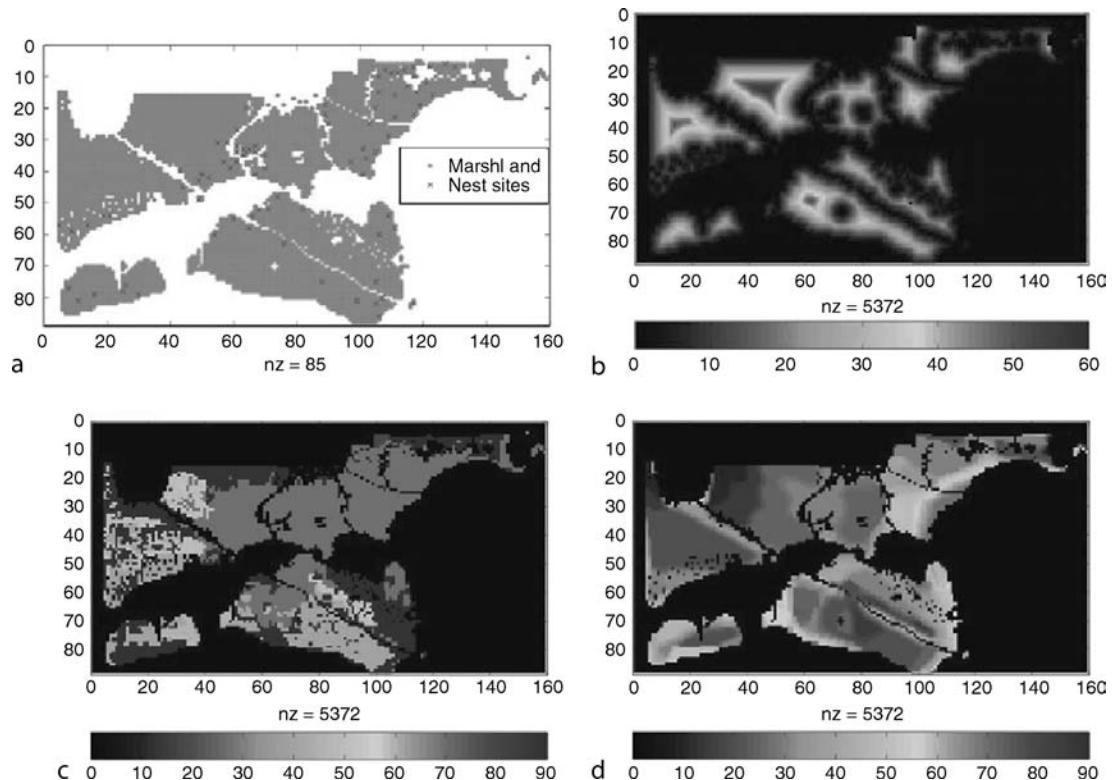
Spatial Data Mining. Figure 2. Spatial outlier (station ID 9) in traffic volume data (best viewed in color).



Spatial Data Mining. Figure 3. An example of spatial co-location patterns.

there show that *trees* and *fire* tend to co-occur together across the spatial region, as well as the pattern *bird* and *house*. Spatial co-location is a generalization of a classical data mining pattern-family called association rules, since transactions are not natural in spatial datasets, and partitioning space across transactions leads to loss of information about neighbor relationships between items near transaction boundaries. Additional details about co-location interest measures, e.g., participation index and K-functions, and mining algorithms are described in [2].

Location prediction is concerned with the discovery of a model to infer preferred locations of a spatial phenomenon from the maps of other explanatory spatial features. For example, ecologists may build models to predict habitats for endangered species using maps of vegetation, water bodies, climate, and other related species. Figure 4 gives an example of a dataset used in building a location prediction model for red-winged



Spatial Data Mining. Figure 4. (a) Learning dataset: The geometry of the Darr wetland and the locations of nests. (b) The spatial distribution of *distance to open water*. (c) The spatial distribution of *vegetation durability* over the marshland. (d) The spatial distribution of *water depth*.

blackbirds in the Darr and Stubble wetlands on the shores of Lake Erie in Ohio, USA. This dataset consists of nest location, distance to open water, vegetation durability and water depth maps. Classical prediction methods may be ineffective in this problem due to the presence of spatial auto-correlation. Spatial data mining techniques that capture the spatial auto-correlation of nest location such as the Spatial Auto-regression Model (SAR) [1] and Markov Random Fields based Bayesian Classifiers (MRF-BC) are used for location prediction modeling. A comparison of these methods is discussed in [6].

Spatial Hotspots are unusual spatial groupings of events that tend to be much more closely related than other events. Examples of spatial hotspots can be incidents of crime in a city or outbreaks of a disease. Hotspot patterns have properties of clustering as well as anomalies from classical data mining. However, hotspot discovery [11] remains a challenging area of research due to variation in shape, size, density of hotspots and underlying space (e.g., Euclidean or spatial networks such as roadmaps). Additional challenges arise from the spatio-temporal semantics such as emerging hotspots, displacement etc.

Key Applications

Spatial data mining and the discovery of spatial patterns has applications in a number of areas. Detecting spatial outliers is useful in many applications of geographic information systems and spatial databases, including the domains of public safety, public health, climatology, and location-based services. As noted earlier, for example, spatial outlier applications may be used to identify defective or out of the ordinary (i.e., unusually behaving) sensors in a transportation system (e.g., Fig. 1). Spatial co-location discovery is useful in ecology in the analysis of animal and plant habitats to identify co-locations of predator-prey species, symbiotic species, or fire events with fuel and ignition sources. Location prediction may provide applications toward predicting the climatic effects of *El Nino* on locations around the world. Finally, identification of spatial hotspots can be used in crime prevention and reduction, as well as in epidemiological tracking of disease.

Cross-references

- ▶ [Data Mining](#)
- ▶ [Geographic Information System](#)

- ▶ [Spatial Network Databases](#)
- ▶ [Spatio-temporal Databases](#)
- ▶ [Spatio-temporal Data Mining](#)

Recommended Reading

1. Cressie N.A. Statistics for Spatial Data (Revised Edition). Wiley, New York, NY, 1993.
2. Huang Y., Shekhar S., and Xiong H. Discovering co-location patterns from spatial datasets: a general approach. *IEEE Trans. Knowl. Data Eng.*, 16(12):1472–1485, 2004.
3. Kou Y., Lu C.T., and Chen D. Algorithms for spatial outlier detection. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 597–600.
4. Longley P.A., Goodchild M., Maquire D.J., and Rhind D.W. *Geographic Information Systems and Science*. Wiley, 2005.
5. Mamoulis N., Cao H., and Cheung D.W. Mining frequent spatio-temporal sequential patterns. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 82–89.
6. Shekhar S., Schrater P., Vatsavai R., Wu W., and Chawla S. Spatial contextual classification and prediction models for mining geospatial data. *IEEE Trans. Multimed.* (special issue on Multimedia Databases), 4(2):174–188, 2002.
7. Shekhar S. and Chawla S. *A Tour of Spatial Databases*. Prentice Hall, 2003.
8. Shekhar S., Lu C.T., and Zhang P. A unified approach to detecting spatial outliers. *GeoInformatica*, 7(2):139–166, 2003.
9. Shekhar S., Zhang P., Huang Y., and Vatsavai R. Trend in spatial data mining. In *Data Mining: Next Generation Challenges and Future Directions*, H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.). AAAI/MIT Press, 2003.
10. Solberg A.H., Taxy T., and Jain A.K. A Markov random field model for classification of multisource satellite imagery. *IEEE Trans. Geosci. Remote Sens.*, 34(1):100–113, 1996.
11. US Department of Justice - Mapping and Analysis for Public Safety report. *Mapping Crime: Understanding Hot Spots*, 2005 (<http://www.ncjrs.gov/pdffiles1/nij/209393.pdf>).

Spatial Data Types

MARKUS SCHNEIDER

University of Florida, Gainesville, FL, USA

Synonyms

- [Geometric data types](#)

Definition

Data types are a well known concept in computer science (for example, in programming languages or in database systems). A *data type* defines a set of homogeneous values and the allowable operations on those values. An example is a type *integer* representing

the set of 32-bit integers and including operations such as addition, subtraction, and multiplication that can be performed on integers. *Spatial data types* or *geometric data types* provide a fundamental abstraction for modeling the geometric structure of objects in space as well as their relationships, properties, and operations. They are of particular interest in *spatial databases* [4,8,12] and *Geographical Information Systems* [4]. One speaks of *spatial objects* as values of spatial data types. Examples are two-dimensional data types for *points* (for example, representing the locations of lighthouses in the U.S.), *lines* (for example, describing the ramifications of the Nile Delta), *regions* (for example, depicting air-polluted zones), *spatial networks* (for example, representing the routes of the Metro in New York), and *spatial partitions* (for example, describing the 50 states of the U.S. and their exclusively given topological relationships of adjacency or disjointedness) as well as three-dimensional data types for *surfaces* (for example, modeling the shape of landscapes) or *volumes* (for example, representing urban areas). Operations on spatial data types include *spatial operations* like the geometric *intersection*, *union*, and *difference* of spatial objects, *numerical operations* like the *length* of a line or the *area* of a region, *topological relationships* checking the relative position of spatial objects to each other like *overlap*, *meet*, *disjoint*, or *inside*, and *cardinal direction relationships* like *north* or *southeast*.

Historical Background

In the late 1970s, the interest to store geometric data into databases arose. The success and efficiency of relational database technology for standard applications, which is rooted in its simple data model, its high-level query languages, and its well understood underlying theory, has led to many proposals to transfer this technology directly to geometric applications and to explicitly model the structure of spatial data as relations (tables). The consequence is that the user conceives spatial data in tabular form, just the same as standard data, and that a spatial object is represented by several or even many tuples. An example of such a relation schema is *RelName(id : integer, x₁ : integer, y₁ : integer, x₂ : integer, y₂ : integer, type : string, <other information>)* where x₁, y₁, x₂, and y₂ are the coordinates of a point or a line segment. The flag *type* indicates whether a tuple describes a point, a single line, a line segment of a line, or a line segment of a polygon. The value *id* denotes the object identifier.

This approach has revealed a number of fundamental drawbacks. Since all lines and polygons are decomposed into a set of line segments (tuples) scattered over a relation, a spatial object is not treated as an entity or unit but only *corresponds* to several tuples. This is different compared to values of standard data types. A second drawback is that the approach forces the user to model complex spatial objects in flat, independent relations. Since the representations of spatial data occurs on a very low level and is exclusively based on standard domains like integers, strings, and reals (while the user has originally intended to deal with points, lines, or polygons (regions)), an adequate treatment of spatial data is impeded. Although the facilities of the query language of a DBMS are available, they are only of limited use. Since such a language is based on standard domains and has no concept of spatial data types, it cannot provide and support any meaningful geometric operations. A more detailed discussion can be found in [9].

Foundations

The numerous deficiencies of the approach of modeling spatial data as relations, have resulted in the assessment that this approach is unsuitable to manage spatial data in a clean and efficient manner, and that a high-level view of spatial objects is essential. This has led to the design of spatial data types that are represented as *abstract data types*, thus provide such a high-level view, and can be used as attribute data types in a database schema in the same way as standard data types like *integer*, *float*, or *string*. That is, the internal structure of a spatial object is hidden from the user, and its features can only be retrieved by (abstract) operations on this object.

One can distinguish different kinds of spatial data types. *Universal spatial data types* either only provide a single generic spatial data type called *spatial*, and therefore do not consider the dimensionality and shape of spatial objects, or they provide the types *spatial_0*, *spatial_1*, *spatial_2*, and *spatial_3* and thus, consider the dimensionality but not the shape of spatial objects [6]. Another conceptual model for spatial data types is based on mathematical abstractions called *point sets*. The user is supplied with the concept that each spatial object consists of an infinite set of points that can be described by finite means. The approach in [7] introduces a type *POINT-SET* for point sets together with a collection of geometric operations. Aspects like

dimensionality and shape of an object are not considered. A further approach of modeling spatial objects is that of using *half planes* [13], where each half plane is defined by a *half plane segment*. A half plane segment uniquely determines a straight line which is given by an inequality, passes this segment and forms the one-sided boundary of a half plane. For constructing a polygonal region, an appropriately arranged sequence of intersection operations (conjunction of inequalities) defined on half planes is employed. This concept is the precursor of so-called *constraint spatial databases*. Most popular and fundamental abstractions of spatial objects fall into the category of *structure-based spatial data types*. These data types organize space into points, lines, regions, surfaces, volumes, spatial partitions, spatial networks, and similarly structured entities. Thus, this approach considers the structural shape and spatial extent of spatial objects, that is, their geometry. Spatial data types for points, lines, and regions have, for example, been considered in [1,5,6,9,11,14], for surfaces and volumes in [10], for spatial partitions in [3], and for spatial networks in [12].

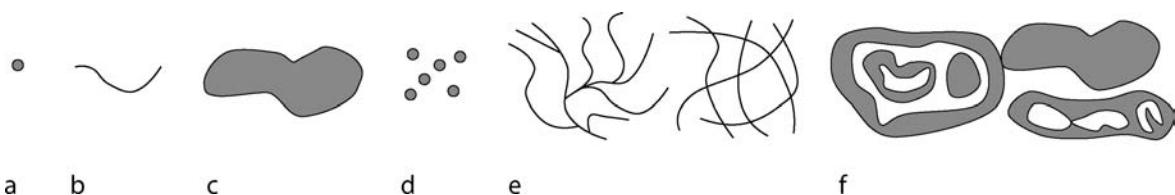
Structure-based spatial data types have prevailed and form the basis of a large number of data models and query languages for spatial data. They have also found broad acceptance in spatial extension packages of commercially and publicly available database systems, as well as in Geographical Information Systems. One can distinguish the older generation of *simple* spatial data types and the newer generation of *complex* spatial data types, depending on the spatial complexity the types are able to model. In the two-dimensional space, simple spatial data types only provide simple object structures like single points, continuous lines, and simple regions (Fig. 1a–c). However, from an application perspective, simple spatial data types have turned out to be inadequate abstractions for spatial applications, since they are insufficient to cope with the variety and complexity of geographic

reality. From a formal perspective, they are not closed under the geometric set operations intersection, union, and difference. This means that these operations applied to two simple spatial objects can produce a spatial object that is not simple. Complex spatial data types solve these problems. They provide universal and versatile spatial objects and are closed under geometric set operations. They allow objects with multiple components, region components that may have holes, and line components that may model ramified, connected geometric networks (Fig. 1d–f).

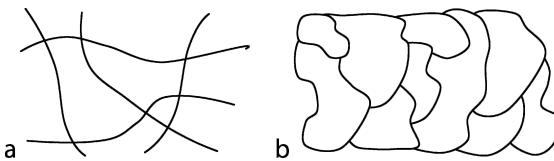
Even more complex structure-based spatial data types are spatial networks and spatial partitions. They are the essential components of maps. A *spatial network* (Fig. 2a) can be viewed as a spatially embedded graph which consists of a set of point objects representing its nodes and a set of line objects describing the geometry of its edges. Examples are highways, rivers, public transport systems, power lines, and phone lines. A *spatial partition* (Fig. 2b) is a set of region objects together with the topological constraint that any two regions either meet or are disjoint. The neighborhood relationship is of particular interest here since region objects may share common boundaries. Examples are states, school districts, crop fields, and land parcels. Both in spatial networks and in spatial partitions, their components (line objects, region objects) are annotated with thematic data like state name, unemployment rate, and parcel id.

Spatial operations manipulate spatial objects. They take spatial objects as operands and return either spatial objects or scalar values (like Boolean or numerical values) as results. One can classify them into the following categories:

Spatial predicates returning Boolean values. A spatial relationship is a relationship between two or more spatial objects. A *spatial predicate* compares two spatial objects with respect to some spatial relationship and



Spatial Data Types. Figure 1. Examples of a simple point object (a), a simple line object (b), a simple region object (c), a complex point object (d), a complex line object (e), and a complex region object (f).



Spatial Data Types. Figure 2. Examples of a spatial network (a) and a spatial partition (b).

thus conforms to a binary relationship returning a Boolean value. Spatial predicates can be classified into three subcategories. *Topological predicates* characterize the relative position of spatial objects towards each other and are preserved under topological transformations such as translation, rotation, and scaling; they do not depend on metric concepts like distance. Examples are the well known predicates *equal*, *disjoint*, *coveredBy*, *covers*, *overlap*, *meet*, *inside*, and *contains* between two simple regions. *Metric predicates* use measurements such as distances. For example, the predicates *in_circle* and *in_window* test if a spatial object is located within the scope of a predefined circle or rectangle. *Directional predicates* like *north* or *southeast* compare the cardinal direction of a target object with respect to a reference object.

Spatial operations returning numbers. These operations compute metric properties of spatial objects and return a number. Examples are the operations *area* and *perimeter* computing the corresponding values of a region object, the operation *length* calculating the total length of a line object, the operation *diameter* determining the largest distance between any two object components, the operation *dist* computing the minimal distance between two spatial objects, and the operation *cardinality* yielding the number of components of a spatial object.

Spatial operations returning spatial objects. These operations return spatial objects as results and can be subdivided into object construction operations, which construct new objects from existing objects, and object transformation operations, which transform one or more spatial objects into a new spatial object. The *object construction operations* include, for example, the *geometric set operations union*, *intersection*, and *difference*, which satisfy closure properties, the operation *convex_hull*, which constructs the smallest convex region (polygon) enclosing a finite collection of points, the operation *boundary*, which returns the boundary of

a region object as a line object, the operation *box*, which determines the minimal, axis-parallel rectangle (called *minimal bounding box* or *rectangle*) that bounds a spatial object, and the operation *components*, which extracts the vertices of a line object. Examples of *object transformation operations* are the operation *extend*, which takes a spatial object *s* and a real number *r* as operands and creates a polygonal region that is a spatial extension of *s* with distance *r* from *s* (also known as *buffer zoning*), the operation *rotate*, which rotates a spatial object around a point, and the operation *translate*, which moves a spatial object by a defined vector. *Spatial operations on spatial networks and spatial partitions.* An important operation on spatial networks is the *shortest_path* operator. It computes the route or path of minimum distance between a source and a destination. An important operation on spatial partitions is the *overlay* operation. It takes two spatial partitions modeling different themes as operands, lays them transparently on top of each other, and combines them into a new spatial partition by intersection. A large collection of other operations is available for both kinds of structures.

A brief example illustrates the embedding of a spatial data type into a relation schema and the posing of a spatial query. Consider the map of the 50 states of the USA. Besides its thematic attributes like name and population, each state is also described by a geometry which is a region. Cities can be represented as points, that is, one is here interested in their location and not so much in their extent. As thematic attributes, one could be interested in their name and population. In the following two relation schemas, the spatial data types *point* and *region* are used in the same way as attribute data types as standard data types.

```
states(sname: string, spop: integer, territory: region)
cities(cname: string, cpop: integer, loc: point)
```

A query could ask for all pairs of city names and state names where a city is located in a state. This can then be formulated as a *spatial join*:

```
select cname, sname
from cities, states
where loc inside territory
```

The term *inside* is a topological predicate testing whether a point object is located inside a region object.

Key Applications

Spatial data types are a universal and general concept for representing geometric information in all kinds of spatial applications. Hence, they are not only applicable to a few key applications. In principle, all applications in the geosciences (for example, geography, hydrology, soil sciences) and Geographical Information Systems, as well as many applications in government and administration (for example, cadastral application, urban planning), can benefit from them. Independent studies have shown that about 80% of all data have spatial features (like geometric attributes) or a spatial reference (like an address). Thus, it is not surprising that independent international studies have predicted that geoinformation technology will belong to the most important and promising technologies in the future, besides biotechnology and nanotechnology.

Cross-references

- ▶ [Cardinal Direction Relationships](#)
- ▶ [Dimension-Extended Topological Relationships](#)
- ▶ [Simplicial Complex](#)
- ▶ [Spatial Operations and Map Operations](#)
- ▶ [Three-Dimensional GIS and Geological Applications](#)
- ▶ [Topological Relationships](#)

Recommended Reading

1. Clementini E. and Di Felice P. A model for representing topological relationships between complex geometric features in spatial databases. *Inf. Syst.*, 90(1–4):121–136, 1996.
2. Egenhofer M.J. Spatial SQL: a query and presentation language. *IEEE Trans. Knowl. Data Eng.*, 6(1):86–94, 1994.
3. Erwig M. and Schneider M. Partition and Conquer. In Proc. Third International Conference on Spatial Information Theory, 1997, pp. 389–408.
4. Güting R.H. An introduction to spatial database systems. *VLDB J.*, 3(4):357–399, 1994.
5. Güting R.H. and Schneider M. Realm-based spatial data types: the rose algebra. *VLDB J.*, 4:100–143, 1995.
6. Güting R.H. Geo-relational algebra: a model and query language for geometric database systems. In *Advances in Database Technology*, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp. 506–527.
7. Manola F. and Orenstein J.A. Toward a general spatial data model for an object-oriented DBMS. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 328–335.
8. Rigaux P., Scholl M., and Voisard A. *Spatial Databases – With Applications to GIS*. Morgan Kaufmann Publishers, 2002.
9. Schneider M. *Spatial Data Types for Database Systems – Finite Resolution Geometry for Geographic Information Systems*, Vol. LNCS 1288. Springer, 1997.

10. Schneider M. and Weinrich B. An abstract model of three-dimensional spatial data types. In Proc. 12th ACM Int. Symp. on Geographic Inf. Syst., 2004, pp. 67–72.
11. Schneider M. and Behr T. Topological relationships between complex spatial objects. *ACM Trans. Database Syst.*, 31(1):39–81, 2006.
12. Shekar S. and Chawla S. *Spatial Databases: A Tour*. Prentice-Hall, 2003.
13. Scholl M. and Voisard A. Thematic map modeling. In Proc. 1st International Symposium on Advances in Spatial Databases, 1989, pp. 167–190.
14. Worboys M.F. and Bofakos P. A canonical model for a class of areal spatial objects. In Proc. Third International Symposium on Advances in Spatial Databases, 1993, pp. 36–52.
15. Worboys M.F. and Duckham M. *GIS: A Computing Perspective*. CRC, 2004.

Spatial Graph Databases

- ▶ [Spatial Network Databases](#)

Spatial Indexing Techniques

YANNIS MANOLOPOULOS¹, YANNIS THEODORIDIS², VASSILIS J. TSOTRAS³

¹Aristotle University of Thessaloniki, Thessaloniki, Greece

²University of Piraeus, Piraeus, Greece

³University of California-Riverside, Riverside, CA, USA

Synonyms

- [Spatial access methods](#)

Definition

A Spatial Index is a data-structure designed to enable fast access to spatial data. Spatial data come in various forms, the most common being points, lines, and regions in n -dimensional space (practically, $n = 2$ or 3 in GIS Geographical Information System applications). Typical “selection” queries include the *spatial range query* (“find all objects that lie within a given query region”), and the *spatial point query* (“find all objects that contain a given query point”). In addition, multi-dimensional data introduce spatial relationships (such as overlapping and disjointness) and operators

(e.g., nearest neighbor), which need to be efficiently supported as well. Example queries are the *spatial join query* (“find all pairs of objects that intersect each other”) and the *nearest neighbor query* (“find the five objects nearest to a given query point”). It should be noted that traditional indexing approaches (B^+ -trees, etc.) are not appropriate for indexing spatial data; the basic reason is the lack of total ordering, which is an inherent characteristic in a multi-dimensional space. As a result, specialized access methods are necessary.

Historical Background

Many applications (VLSI, CAD/CAM, GIS, multimedia) need to represent, store and manipulate spatial data types, such as points, lines, and regions in n -dimensional space. Although the representation of this type of data may be straightforward in a traditional database system (e.g., a 2-dimensional point may be represented as a pair of x - and y - numeric values), spatial relationships (e.g., overlapping) and operators (e.g., nearest neighbor) need to be efficiently supported as well. These spatial relationships and operators have led to a variety of interesting and more complex queries like spatial joins, nearest neighbors etc. As a result, specialized access methods have been proposed in order to quickly answer the above complex queries, as well as spatial range/point queries.

Given the characteristics of spatial data, for each spatial operator the query object’s geometry needs to be combined with each data object’s geometry. Nevertheless, the processing of complex geometry representations, usually polygons, is very expensive in terms of CPU cost. For that reason, the object geometries are approximated (typically by Minimum Bounding Rectangles –MBRs), and these approximations are then stored in underlying indices while the actual geometry is stored separately. As a result, a two-step procedure is involved during query processing, consisting of a *filter step* and a *refinement step*. The question that arises is how the object approximations (MBRs) are organized in order to answer the hits and the candidates, i.e., the result of the filter step.

Various spatial indices have been proposed in the literature and can be divided in two categories: indices designed for multi-dimensional points, and indices for multi-dimensional regions. Examples in the first category are the LSD tree [7], the Grid File [10], the hB-tree [9], the Buddy Tree [14] and the BV-tree [3]. The major

representatives in second category are the R-tree [5] and the Quadtree [2], and their variants.

Given the complexity of the indexing problem and the different requirements of the multiple applications that index spatial data, it is not clear which the best index is. Nevertheless, R-tree implementations have found their way into commercial DBMSs. This is mainly due to their simplicity and ease of implementation (their structure is an adaptation of the B^+ -tree for spatial data), as well as their robust performance for many applications.

Foundations

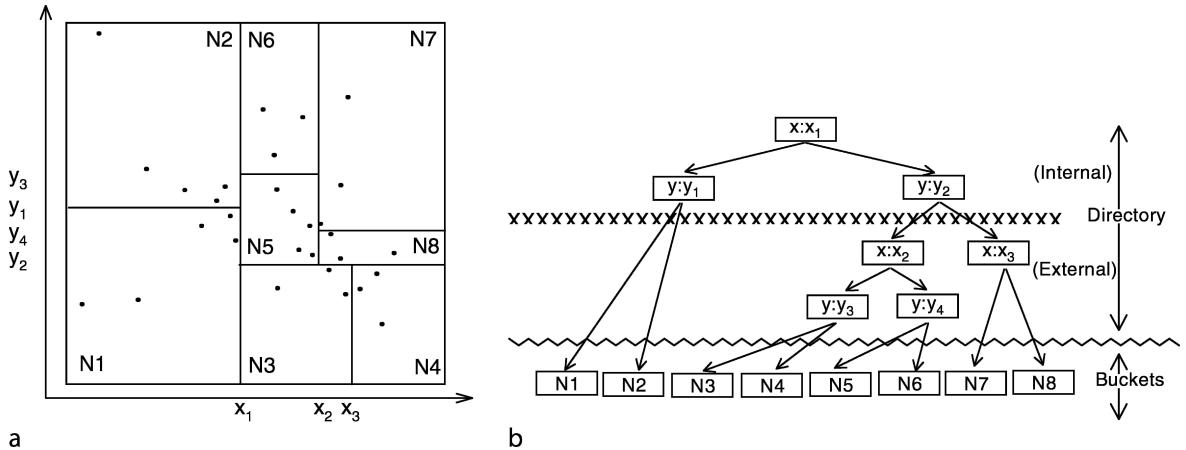
This section first discusses indices for multi-dimensional points, while the description of major indices for multi-dimensional (non-point) regions follows.

Indices for Multi-Dimensional Points

The LSD-tree (Local Split Decision tree), proposed in [1], maintains a catalog that separates space in a collection of (non-equal sized) disjoint subspaces using the extended k-d tree structure. New entries are inserted into the appropriate bucket. When an overflow happens then the bucket is split and the information about the partition line (split dimension and split position) is stored in a directory. Thus the overall structure of the LSD-tree consists of data buckets and a directory tree. The directory tree is kept in main memory until it grows more than a threshold; then a sub-tree is stored in an external catalogue in order for the whole structure to remain balanced (an example appears in Fig. 1). Inserting a new entry (point) in the LSD-tree is straightforward since nodes are disjoint. However, the target node may overflow due to an insertion; a split procedure then takes place.

The LSD-tree is a *space-driven structure*, i.e., it decomposes the complete workspace. Other members of this family include the Grid File [10] and the hB-tree [9]. On the other hand, *data-driven structures* only cover those parts of the workspace that contain data objects. Examples are the Buddy Tree [14] and the BV-tree [3].

The Grid File is an access method comprising of two separate parts: (i) the *directory*, and (ii) the *linear scales*. The Grid File imposes a grid on the indexed multidimensional attribute space. Each cell in this grid corresponds to one data page. The data points that “fall” inside a given cell are stored in the cell’s corresponding page. Each cell must thus store a



Spatial Indexing Techniques. Figure 1. The LSD-tree.

pointer to its corresponding page. This information is stored in the Grid File's *directory*. The information of how each dimension is divided (and thus how data values are assigned to cells) is kept in the *linear scales*. The Grid File can be thought as a multidimensional extension of hashing. As a result, exact match queries take only two disk accesses one for the directory and one for the data page.

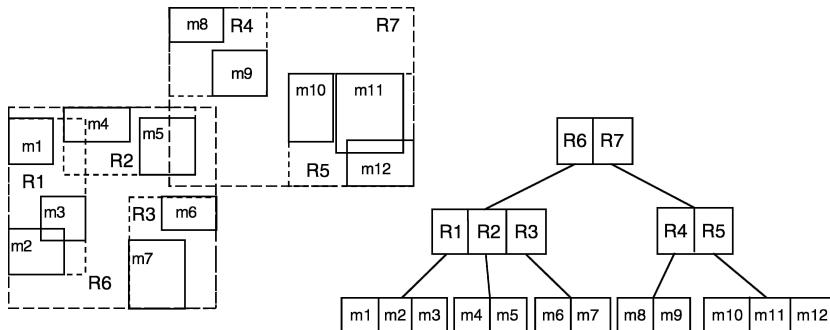
Indices for Multi-Dimensional Regions

As with point indexing, two different approaches (data-driven and space-driven) have been proposed for indexing regions as well. The main representatives are the R-tree [5] and the Quadtree, [2,13], which were later followed by dozens of variants. In the sequel, the two structures are presented in detail. The reader is referred to a recent exhaustive survey [4] for further reading on their variants.

R-trees were originally proposed [5] as a direct extension of B⁺-trees in n -dimensional space. The data structure is a height-balanced tree that consists of intermediate and leaf nodes. A leaf node is a collection of entries of the form (o_id, R) where o_id is an object identifier, used to refer to an object in the database, and R is the MBR minimum bounding rectangle approximation of the data object. An intermediate node is a collection of entries of the form (ptr, R) where ptr is a pointer to a lower level node of the tree and R is a representation of the minimum rectangle that encloses all MBRs of the lower-level node entries. Let M be the maximum number of entries in a node and let $m \leq M/2$ be a parameter specifying the minimum number of entries in a node. An R-tree satisfies

the following properties: (i) every leaf node contains between m and M entries unless it is the root; (ii) for each entry (o_id, R) in a leaf node, R is the MBR minimum bounding rectangle approximation of the object represented by o_id ; (iii) every intermediate node has between m and M children unless it is the root; (iv) for each entry (ptr, R) in an intermediate node, R is the smallest rectangle that completely encloses the rectangles in the child node; (v) the root node has at least two children unless it is a leaf; and (vi) all leaves appear at the same level. As an example, Fig. 2 illustrates several MBRs minimum bounding rectangle m_i and the corresponding R-tree built on these rectangles (assuming maximum node capacity $M = 3$).

In order for a new entry E to be inserted into the R-tree, starting from the root node, the child that needs minimum enlargement to include E is chosen (ties are resolved by choosing the one with the smallest area). When a leaf node N is reached, E is inserted into that, probably causing a split if N is already full. In such a case, the existing entries together with E are redistributed in two nodes (the current and a new one) with respect to the minimum enlargement criterion. In the original paper [6] three alternatives were proposed in order to find the two groups: an exhaustive, a quadratic-cost, and a linear-cost split algorithm. The processing of a point or range query with respect to a query window q (which could be either point or rectangle, respectively) is straightforward: starting from the root node, several tree nodes are traversed down to the leaves, depending on the result of the overlap operation between q and the corresponding node rectangles. When the search algorithm reaches



Spatial Indexing Techniques. Figure 2. The R-tree.

the leaf nodes, all data rectangles that overlap the query window q are added to the answer set. Regarding k -nearest-neighbor queries, [12] proposed customized branch-and-bound algorithms for R-trees.

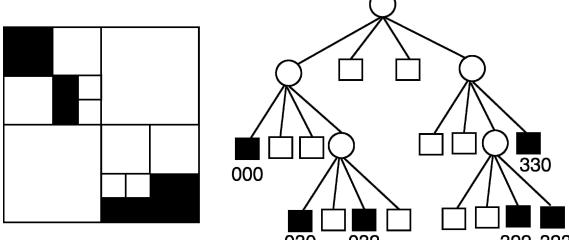
After Guttman's proposal, several researchers proposed their own improvements on the basic idea. Roussopoulos and Leifker [11] proposed the *Packed* R-tree for bulk loading data in an R-tree. Objects are first sorted in some desirable order (according to the low- x value, low- y value, etc.) and then the R-tree is bulk loaded from the sorted file and R-tree nodes are packed to capacity. Note that the above techniques allow node "overlapping": MBRs of different nodes can overlap. Since no disjointness is guaranteed, during a search multiple paths of the R-tree may be traversed. An efficient variation, namely the R^+ -tree, was proposed by Sellis et al. [15]. To preserve disjointness among node rectangles, the R^+ -tree uses a "clipping" technique that duplicates data entries when necessary. However, the penalty is a (possibly high) increase in space demand due to the replication of data, which, in turn, degenerates search performance. Generally speaking, clipping techniques are ideal for point queries because a single path should be traversed, while range queries tend to be expensive, when compared with the overlapping techniques.

Later, Beckman et al. [1] and Kamel and Faloutsos [8] proposed two R-tree-based methods, the R^* -tree and the Hilbert R-tree, respectively, which are currently considered to be the most efficient members of the R-tree family in terms of query performance. The R^* -tree uses a rather complex but more effective grouping algorithm to split nodes by computing appropriate area, perimeter, and overlap values while the Hilbert R-tree actually stores Hilbert values at the leaf level and ranges of those values at the upper levels, similarly

to the B^+ -tree construction algorithm. In addition, a "lazy" split technique is followed, where overflow entries are evenly distributed among sibling nodes and only when all those are full, a new node (hence, split) is created.

The *Region Quadtree* [2] is the most popular member in the Quadtree family. It is used for the representation of binary images, that is $2^n \times 2^n$ binary arrays (for a positive integer n), where a "1" ("0") entry stands for a black (white) picture element. More precisely, it is a degree four tree with height n , at most. Each node corresponds to a square array of pixels (the root corresponds to the whole image). If all of them have the same color (black or white) the node is a leaf of that color. Otherwise, the node is colored grey and has four children. Each of these children corresponds to one of the four square sub-arrays to which the array of that node is partitioned. It is assumed here, that the first (leftmost) child corresponds to the upper left sub-array, the second to the upper right sub-array, the third to the lower left sub-array and the fourth (rightmost) child to the lower right sub-array, denoting the directions NW, NE, SW, SE single ended, respectively. Figure 3 illustrates a Quadtree for an 8×8 pixel array. Note that black (white) squares represent black (white) leaves, whereas circles represent internal nodes (also, grey ones).

Region Quadtrees, as presented above, can be implemented as main memory tree structures (each node being represented as a record that points to its children). Variations of Region Quadtrees have been developed for secondary memory. *Linear Region Quadtrees* [8] are the ones used most extensively. A linear Quadtree representation consists of a list of values where there is one value for each black node of the pointer-based Quadtree. The value of a node is an address describing



Spatial Indexing Techniques. Figure 3. The Quadtree.

the position and size of the corresponding block in the image. These addresses can be stored in an efficient structure for secondary memory (such as a B⁺-tree). There are also variations of this representation where white nodes are stored too, or variations which are suitable for multicolor images. Evidently, this representation is very space efficient, although it is not suited to many useful algorithms that are designed for pointer-based Quadtrees. The most popular linear implementations are the FL Fixed Length (Fixed Length), the FD Fixed Depth (Fixed length – Depth) and the VL Variable Length (Variable Length) linear implementations [13]. Techniques for computing various kinds of geometric properties have also been developed. Connected component labeling, polygon coloring and computation of various types of perimeters fall in this category. Finally, many operations on images have been developed. For example, point location, set operations on two or more images (intersection, union, difference, etc.), window clipping, linear image transformations and region expansion.

Other region Quadtree variants have appeared in the literature mainly for indexing non-regional data. MX Quadtrees are used for storing points seen as black pixels in a Region Quadtree. PR Quadtrees are also used for points. However, points are drawn from a continuous space, in this case. MX-CIF Quadtrees are used for small rectangles. Each rectangle is associated with the Quadtree node corresponding to the smallest block that contains the rectangle. PMR Quadtrees are used for line segments. Each segment is stored in the nodes that correspond to blocks intersected by the segment. A detailed presentation of these and other region Quadtree variants is given in [8].

Key Applications

Geographic Information Systems (GIS) deal extensively with the management of 2- and 3- dimensional spatial

data. For example, a map typically contains point objects (locations of interest), line objects (road segments, highways, rivers, etc.) as well as region objects (lakes, forests, etc.) GIS use spatial indexing as a means to provide fast access to large amounts of spatial data.

Multimedia Systems manage multimedia objects like images, text, audio, video, etc. A typical query in such systems is the *similarity* query (i.e., find objects that are similar to a query object according to some measure). To answer these queries, each multimedia object is abstracted by a set of multidimensional points (features). These multidimensional points are then indexed by a spatial index. Similarly, the query object is represented by a multidimensional point. The similarity query is then answered as a nearest neighbor query (i.e., find the nearest neighbor(s) to the point that represents the query).

The World-Wide Web has also provided new applications for geographic related queries and thus spatial indexing. Users can now find maps, driving directions, etc. through specialized web sites that typically offer the ability to perform spatial queries.

Location-based services provide querying capabilities based on the location of the user (e.g., “find the cheapest gas station within 5 miles of my car”). As the user moves in space, the results of the queries change. Such queries typically have a spatial component and spatial (and spatio-temporal) indexes are used to provide fast response.

CAD systems use spatial objects to store surfaces and bodies of design objects (for e.g., the wings or the wheels of an airplane). Typical spatial queries involve the proximity of spatial objects, their overlap etc. Related queries (but mainly in the 2-dimensional space) are also relevant for VLSI design systems; here the layout of a chip involves various rectangular regions and overlap and proximity queries are of importance.

Computer Games also involve many spatial searches. In such an environment, players move around in a 3-dimensional space and need to be able to see parts of (partially hidden) objects, various triggers are initiated if a player passed over them, or an explosion needs to identify the nearby objects that are affected. Spatial indexing is used to improve such query response.

Medical Imaging also involves large amounts of 2- and 3-dimensional spatial data. Consider for example X-rays, or, magnetic resonance imaging (MRI)

brain scans. Again, proximity, overlap and related spatial queries are of interest.

Experimental Results

In general, for every presented method, there is an accompanying experimental evaluation in the corresponding reference.

Data Sets

A large collection of real spatial datasets, commonly used for experiments, can be found at *R-tree-portal* (URL: <http://www.rtreeportal.org/>).

URL to Code

R-tree portal (see above) contains the code for most common spatial and spatio-temporal indexes, as well as data generators and several useful links for researchers and practitioners in spatio-temporal databases. Similarly, the Spatial Index Library [6] provides a general framework for developing various spatial indices (URL: <http://dblab.cs.ucr.edu/spatialindexlib>).

Cross-references

- ▶ [B-Tree](#)
- ▶ [GIS](#)
- ▶ [Grid File \(and family\)](#)
- ▶ [Nearest Neighbor Query](#)
- ▶ [Quadtrees \(and family\)](#)
- ▶ [R-Tree \(and family\)](#)
- ▶ [Spatial Join](#)

Recommended Reading

1. Beckmann N., Kriegel H.-P., Schneider R., and Seeger B. The R*-tree: an efficient and robust access method for points and rectangles. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 322–331.
2. Finkel R.A. and Bentley J.L. Quad Trees: a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.
3. Freeston M.A. General solution of the n-dimensional B-tree problem. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp 80–91.
4. Gaede V. and Guenther O. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.
5. Guttman A. R-trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
6. Hadjieleftheriou M., Hoel E., and Tsotras V.J. Sail: A spatial index library for efficient application integration. *GeoInformatica*, 9(4):367–389, 2005.
7. Henrich A., Six H.-W., and Widmayer P. The LSD tree: spatial access to multidimensional point and non point objects. In Proc. 15th Int. Conf. on Very Large Data Bases, 1989, pp. 43–53.
8. Kamel I. and Faloutsos C. Hilbert R-tree: an improved R-tree using fractals. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 500–509.
9. Lomet D.B. and Salzberg B. The hB-tree: a multiatribute indexing method with good guaranteed performance. *ACM Trans. Database Syst.*, 15(4):625–658, 1990.
10. Nievergelt J., Hinterberger H., and Sevcik K.C. The grid file: an adaptable symmetric multikey file structure. *ACM Trans. Database Syst.*, 9(1):38–71, 1984.
11. Roussopoulos N. and Leifker D. Direct Spatial Search on Pictorial Databases Using Packed R-trees. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1985, pp. 17–31.
12. Roussopoulos N., Kelley S., and Vincent F. Nearest neighbor queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 71–79.
13. Samet H. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1990.
14. Seeger B. and Kriegel H.-P. The Buddy-tree: an efficient and robust access method for spatial database systems. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 590–601.
15. Sellis T., Roussopoulos N., and Faloutsos C. The R⁺-tree: a dynamic index for multidimensional objects. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 507–518.

-

Spatial Information System

- ▶ [Geographic Information System](#)
- ▶ [Three-Dimensional GIS and Geological Applications](#)

Spatial Join

NIKOS MAMOULIS

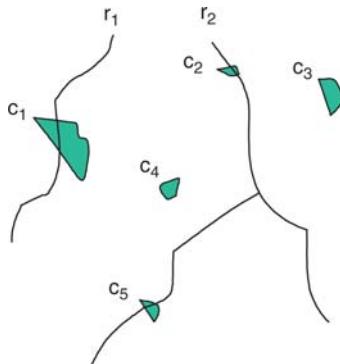
University of Hong Kong, Hong Kong, China

S

Definition

The spatial join is one of the core operators in spatial database systems. Efficient spatial join evaluation is important, due to its high cost compared to other queries, like spatial selections and nearest-neighbor searches. A binary (i.e., pairwise) spatial join combines two datasets with respect to a spatial predicate (usually overlap/intersect). A typical example is “find all pairs of cities and rivers that intersect.” For instance, in Fig. 1 the result of the join between the set of cities {c₁, c₂, c₃, c₄, c₅} and rivers {r₁, r₂}, is {(r₁, c₁), (r₂, c₂), (r₂, c₅)}.

The query in this example is a spatial intersection join. In the general case, the join predicate could be a combination of topological, directional, and



Spatial Join. Figure 1. Graphical example of a spatial intersection join.

distance spatial relations. Apart from the intersection join, variants of the distance join have received considerable attention, because they find application in data analysis tasks (e.g., data mining, clustering). Given two sets R and S of spatial objects (or multidimensional points), and a distance function $dist()$, the e -distance join (or else similarity join [1]) returns the pairs of objects $\{(r, s) : r \in R, s \in S, dist(r, s) \leq e\}$. A closest pairs query [2] returns the set of closest pairs $CP = \{(r, s) : r \in R, s \in S\}$, such that $dist(r, s) \leq dist(r', s')$, for all $r' \in R, s' \in S : (r', s') \notin CP$.

Historical Background

The first spatial join methods [3,4] assume that both inputs are indexed by some spatial access method (e.g., R-trees). The latest spatial join techniques do not rely on pre-existing indices [5–11]. Such situations may arise when at least one input is an intermediate result of a preceding operator. Consider for instance the query “find all rivers of width larger than 20 m, which intersect a forest.” If there is a large percentage of narrow rivers, it might be natural to process the selection part of the query before the spatial join. In such an execution plan, even if there exists a spatial index on rivers, it is not employed by the join algorithm.

Table 1 classifies the spatial join techniques according to the assumption they make on pre-existing indices for the joined inputs. Methods of the first column can be applied only when both inputs are indexed (e.g., two relations Forests and Rivers are joined with respect to a spatial predicate). The second column includes algorithms suitable when only one input is indexed by an R-tree (e.g., Forests are joined with Rivers wider than 20 m). Join algorithms in the

Spatial Join. Table 1. Classification of spatial join methods

Both inputs are indexed	One input is indexed	Neither input is indexed
<ul style="list-style-type: none"> Transformation to z-values and use of B-trees [4] 	<ul style="list-style-type: none"> Indexed nested loops 	<ul style="list-style-type: none"> Spatial hash join [7]
<ul style="list-style-type: none"> Synchronized tree traversal [3] 	<ul style="list-style-type: none"> Seeded tree join [8] 	<ul style="list-style-type: none"> Partition-based spatial merge join [11]
	<ul style="list-style-type: none"> Build a second R-tree and match it with the existing [10, 11] 	<ul style="list-style-type: none"> Size separation spatial join [6]
	<ul style="list-style-type: none"> Sort and match [10] 	<ul style="list-style-type: none"> Sweeping-based spatial join [5]
	<ul style="list-style-type: none"> Slot-index spatial join [9] 	

last column can be used in cases when both inputs are not indexed (e.g., Forests that intersect some City are joined with Rivers wider than 20 m). Most of the spatial join techniques focus on the filter step of the query. The refinement step (i.e., testing the exact geometry of objects against the join predicate) is applied independently of the algorithm, used for the filter step to the pairs that pass it, afterwards.

Foundations

Early Spatial Join Algorithms

Most early spatial join algorithms apply transformation of objects in order to overcome difficulties due to their spatial extent and dimensionality. The first known spatial join algorithm [4] uses a grid to regularly divide the multidimensional space into small blocks, called pixels, and uses a space-filling curve (z-ordering) to order them. Each object is then approximated by the set of pixels intersected by its MBR, i.e., a set of z-values. Since z-values are one-dimensional, the objects can be dynamically indexed using relational index structures like the B^+ -tree. The spatial join is then performed in a sort-merge fashion. The performance of the algorithm depends on the granularity of the grid; larger grids

can lead to finer object approximations, but also increase the space requirements. Other approaches transform the MBRs of the objects into higher dimensional points and use k-d-trees or grid-files to index the points. The join is then performed by the use of these data structures in a similar way as relational multi-attribute joins.

The R-Tree Join R-tree Join (RJ) [2], often referred to as *tree matching* or *synchronous traversal*, computes the spatial join of two relations provided that they are both indexed by R-trees [5]. RJ synchronously traverses both trees, starting from the roots and following entry pairs which intersect. Let n_R, n_S be two directory (non-leaf) nodes of the R-trees that index relations R and S , respectively. RJ is based on the following observation: if two entries $e_i \in n_R$ and $e_j \in n_S$ do not intersect, there can be no pair (o_R, o_S) of intersecting objects, where o_R and o_S are under the sub-trees pointed by e_i and e_j , respectively. A simple pseudo-code for RJ that outputs the result of the filter spatial join step (i.e., outputs pairs of objects whose MBRs intersect) is given in Fig. 2. The pseudo-code assumes that both trees

```

function RJ(Node  $n_R$ , Node  $n_S$ )
  for each  $e_i \in n_R$ 
    for each  $e_j \in n_S$ , such that  $e_i.\text{MBR} \cap e_j.\text{MBR} \neq \emptyset$ 
      if  $n_R$  is a leaf node then /*  $n_S$  is also a leaf node */
        output ( $e_i.\text{ptr}, e_j.\text{ptr}$ ); /* a pair of object-ids
          passing the filter step */
      else /*  $n_R, n_S$  are directory nodes */
        RJ( $e_i.\text{ptr}, e_j.\text{ptr}$ ); /* run recursively for the nodes
          pointed by intersecting entries */
    
```

Spatial Join. Figure 2. The R-tree Join (RJ) Algorithm.

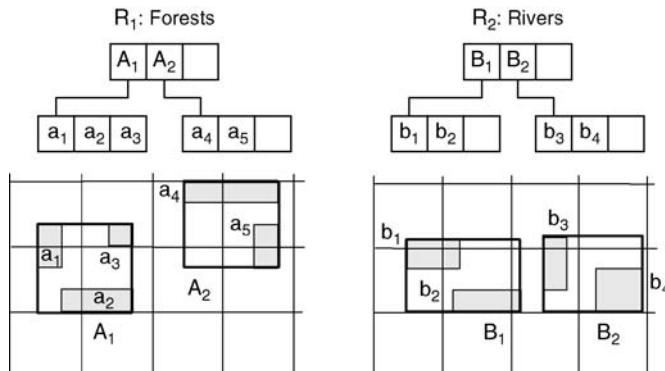
have the same height, yet it can be easily extended to the general case by applying range queries to the deeper tree when the leaf level of the shallow tree is reached.

Figure 3 illustrates two datasets indexed by R-trees. Initially, RJ is run taking the tree roots as parameters. The qualifying entry pairs at the root level are (A_1, B_1) and (A_2, B_2) . Notice that since A_1 does not intersect B_2 , there can be no object pairs under these entries that intersect. RJ is recursively called for the nodes pointed by the qualifying entries until the leaf level is reached, where the intersecting pairs (a_1, b_1) and (a_2, b_2) are output.

Two optimization techniques can be used to improve the CPU speed of RJ [2]. The first (search space restriction) reduces the quadratic number of pairs to be evaluated when two nodes n_R, n_S are joined. If an entry $e_R \in n_R$ does not intersect the MBR of n_S (that is the MBR of all entries contained in n_S), then there can be no entry $e_S \in n_S$, such that e_R and e_S overlap. Using this fact, space restriction performs two linear scans in the entries of both nodes before RJ, and prunes out from each node the entries that do not intersect the MBR of the other node. The second technique, based on the plane sweep paradigm [15], applies sorting in one dimension in order to reduce the cost of computing overlapping pairs between the nodes to be joined. Plane sweep also saves I/Os compared to nested loops, because consecutive computed pairs overlap with high probability.

Algorithms That Do Not Consider Indexes

The most straightforward and intuitive algorithm that can be used to join two relations that are not indexed is the Nested Loops Join. This method can be applied for any type of joins (spatial, non-spatial) and condition predicates (topological, directional, distance, etc.).



Spatial Join. Figure 3. Two datasets indexed by R-trees.

On the other hand, nested loops is the most expensive algorithm, since its cost is quadratic to the size of the relations (assuming that R and S have similar sizes). In fact, evaluation can be performed much faster. Spatial join algorithms for non-indexed inputs process the join in two steps; first the objects from both inputs are preprocessed in some data structures, and then these structures are used to quickly match objects that cover the same area. The algorithms differ in the data structure they use and the way the data are preprocessed.

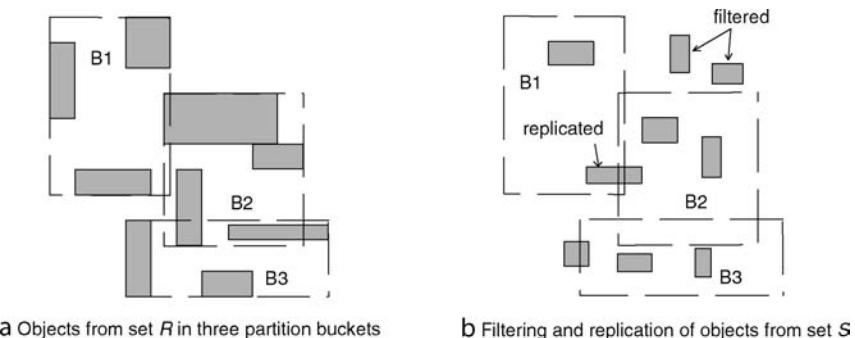
Spatial Hash Join The Spatial Hash Join (HJ) [9] has common features with the relational hash-join algorithm. Set R is partitioned into K buckets, where K is decided by system parameters, such that the expected number of objects hashed in a bucket will fit in memory. The initial extents of the buckets are determined by sampling. Each object is inserted into the bucket whose bounding box is enlarged the least after the insertion. Set S is hashed into buckets with the same extent as R 's buckets, but with a different insertion policy; an object is inserted into all buckets that intersect it. Thus, some objects may go into more than one bucket (*replication*), and some may not be inserted at all (*filtering*). The algorithm does not ensure partitions of equal number of objects from R , as sampling cannot guarantee the best possible slots. Equal sized partitions for S cannot be guaranteed in any case, because the distribution of the objects in the two datasets may be totally different. Figure 4 shows an example of two datasets, partitioned using HJ.

After hashing set S into buckets, the two bucket sets are joined; each bucket B_i^R from R is matched with the corresponding bucket B_i^S from S that covers the same

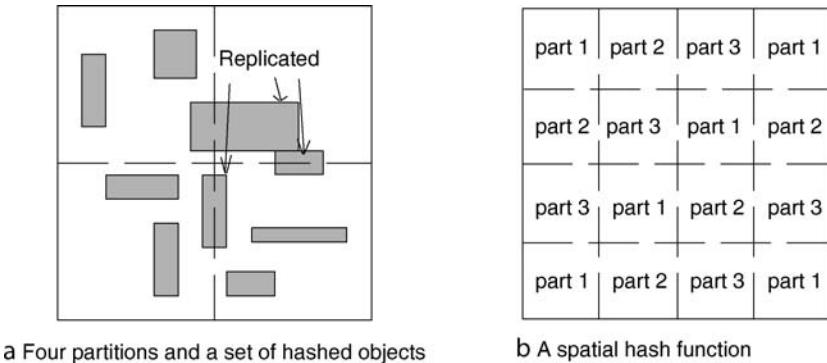
spatial region. For this phase, a single scan of both sets of buckets is required, unless for some pairs of buckets none of them fits in memory. If one bucket fits in memory, it is loaded and the objects of the other bucket are matched with it in a nested-loops fashion. If none of the buckets fits in memory, an R-tree is dynamically built for one of them, and the bucket-to-bucket join is executed in an indexed nested-loop fashion.

Partition Based Spatial Merge Join Partition-based Spatial Merge Join (PBSM) [14] is also based on the hash join paradigm. The space, in this case, is regularly partitioned using an orthogonal grid, and objects from both datasets are hashed into partitions corresponding to grid cells, replicating wherever necessary. Figure 5a illustrates a regular space partitioning incurred by PBSM and some data hashed into the partitions. Objects hashed into the same partitions are then joined in memory using plane sweep. If the data inserted in a partition do not fit in memory, the algorithm recursively repartitions the cell into smaller parts and redistributes the objects. Since data from both datasets may be replicated, the output of the algorithm has to be sorted in order to remove pairs reported more than once.

When the data to be joined are skewed, some partitions may contain a large percentage of the hashed objects, whereas others very few objects, rendering the algorithm inefficient. In order to evenly distribute the data in the partitions and efficiently handle skewed data, a spatial hash function is introduced. The cells of the grid are assigned to partitions according to this function and the space covered by a partition is no longer continuous, but consists of a number of scattered tiles. Figure 5b shows such a (round-robin like) spatial hash function.



Spatial Join. Figure 4. The partitioning phase of HJ algorithm.

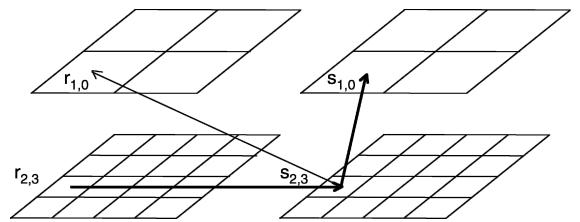


Spatial Join. Figure 5. Regular partitioning by PBSM.

Size Separation Spatial Join Another algorithm that applies regular partitioning, like PBSM, but avoids object replication is Size Separation Spatial Join (S^3J) [6]. S^3J uses a hierarchical space decomposition. L partition layers of progressively larger resolution are introduced; the layer at level l partitions the space into $4l$ cells. A rectangle is then assigned to the topmost layer where it is not intersected by a grid line. This method achieves separation of the data according to their size. The rectangles in each layer are then sorted according to the Hilbert value of their MBRs center. A synchronized scan of the layer files is finally performed and the rectangles from dataset R in a partition at level l are joined with all partitions of dataset S that intersect it at levels $0, \dots, l$. A partition from S is joined with partitions from R at levels $0, \dots, l - 1$. The Hilbert values of the data inside a layer determine the order of the join, avoiding scanning a partition more than once. Figure 6 shows two partition layers of both datasets. Partition $r_{2,3}$ is joined with $s_{2,3}$ and $s_{1,0}$, and partition $s_{2,3}$ is joined with $r_{1,0}$.

S^3J also maintains a dynamic spatial bitmap which, after partitioning the first set, indicates the cells at each layer that contain at least one rectangle, or cover same area with cells at other layers that contain at least one rectangle. This bitmap can be used during the partitioning of the second set to filter entries that cannot intersect any rectangle of the first set. If a rectangle from set S is to be hashed into a partition cell and the bitmap entry of the cell is zero, the hashed rectangle is filtered out.

Scalable Sweeping-Based Spatial Join The Scalable Sweeping-based Spatial Join (SSSJ) [1] is a relatively simple algorithm that is based on plane sweep. Both datasets are sorted according to the lower bound of



Spatial Join. Figure 6. Size separation spatial join.

their projection on an axis (e.g., the x -axis), and some variant of plane sweep (e.g., the *forward-sweep* algorithm described before) is applied to compute the intersection pairs. SSSJ is based on the square-root rule: the expected number of rectangles in a dataset R that intersect the sweep line is $\sqrt{|R|}$, where $|R|$ is the total number of rectangles in R . SSSJ initiates an internal memory plane sweep algorithm. If it runs out of memory, i.e., the rectangles intersected by the sweep line do not fit in memory, the space is dynamically partitioned by stripes parallel to the sorted axis, the rectangles are hashed into the stripes, and plane sweep is recursively executed for each stripe.

Single-Index Join Methods

Methods in this class can be applied when one input is not indexed. Such situations often arise when processing complex queries, where another operator precedes the spatial join. Notice that in this case index-based methods cannot directly be applied, because the intermediate result is not supported by any index. Also, algorithms that consider non-indexed inputs could be expensive. All single-index join methods were proposed after RJ, and they assume that the indexed input is supported by an R-tree. Most of them build a

second structure for the non-indexed input and match it with the existing tree.

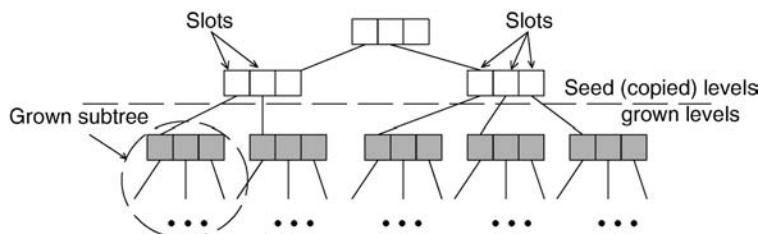
Indexed Nested Loops Join In accordance to the equivalent algorithm for relational joins, the Indexed Nested Loops Join (INLJ) applies a window query to the existing R-tree for each rectangle from the non-indexed set. This method can be efficient only when the non-indexed input is very small. Otherwise, the large number of selection queries can incur excessive computational overhead and access a large number of index pages.

Seeded Tree Join Let R be a dataset indexed by an R-tree and S be a non-indexed dataset. The Seeded Tree Join algorithm (STJ) [10] builds an R-tree for S , using the existing for R as a seed, and then applies RJ to match them. The rationale behind creating a *seeded* R-tree for the second input, instead of a normal R-tree, is the fact that if the new tree has similar high-level node extents with RA, this would lead to minimization of overlapping node pairs during tree matching. Thus, the seeded tree construction algorithm creates an R-tree which is optimal for the spatial join and not for range searching. The seeded tree construction is divided into two phases: the *seeding* phase and the *growing* phase. At the seeding phase, the top k levels (k is a parameter of the algorithm) of the existing R-tree are copied to formulate the top k levels of the new R-tree for S . The entries in the lowest of these levels are called slots. After copying, the slots maintain the copied extent, but they point to empty (null) sub-trees. During the growing phase, all objects from S are inserted into the seeded tree. A rectangle is inserted under the slot that contains it, or needs the least area enlargement. Figure 7 shows an example of a seeded tree structure. The top $k = 2$ levels of the existing R-tree are copied to guide the insertion of the second dataset.

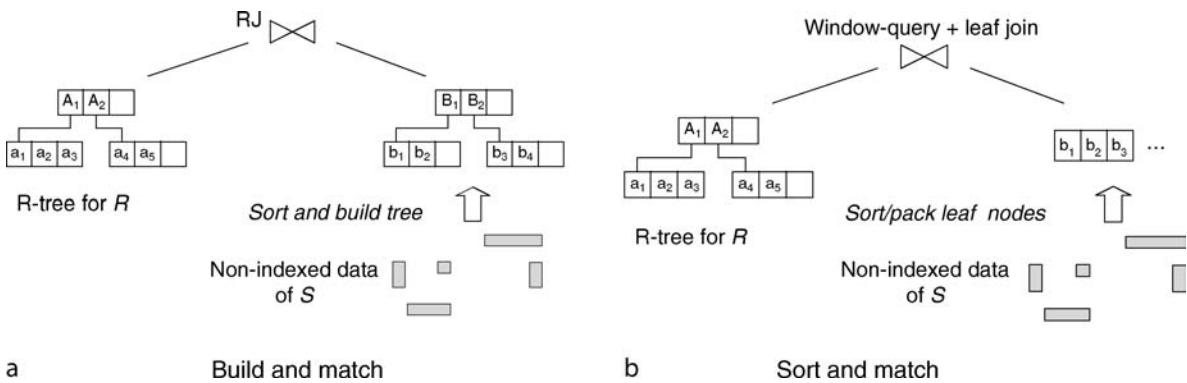
Build and Match Building a packed R-tree using bulk loading can be much more efficient in terms of both CPU time and I/O than constructing it incrementally. Moreover, packed R-trees have a minimum number of nodes and height, and could be very efficient for range queries and spatial joins. The Build and Match (BaM) method [13,14] first builds a packed R-tree for the non-indexed dataset S and then joins it with the existing tree of R , using RJ.

Sort and Match Sort and Match (SaM) [13] is an alternative of BaM, which avoids building a whole R-tree structure prior to matching. The algorithm employs an R-tree bulk-loading technique [8] to sort the rectangles from the non-indexed dataset S but, instead of building the packed tree, it matches each in-memory created leaf node with the leaf nodes from the R-tree of R that intersect it, using the structure of the tree to guide search. For each produced leaf node n_L at the last phase of STR, a window query using the MBR of n_L is applied on R 's tree, in order to identify the leaf nodes there that intersect n_L . Plane sweep is then applied to match n_L with the qualifying leaves of R 's tree. The matching phase of SaM is expected to be efficient, as two consecutive produced nodes will be close to each other with high probability, and there will be good utilization of the LRU buffer. Graphical examples of BaM and SaM are shown in Fig. 8.

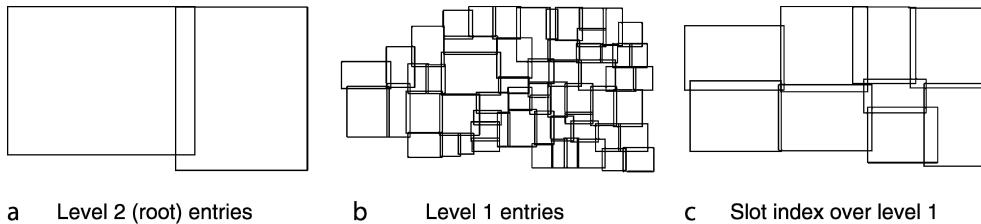
Slot Index Spatial Join The Slot Index Spatial Join [9] is a hash-based spatial join algorithm, appropriate for the case where only one of the two joined relations is indexed by an R-tree. It uses the existing R-tree to define a set of hash buckets. If K is the desired number of partitions (tuned according to the available memory), SISJ will find the topmost level of the tree such that the number of entries there is larger than or equal to K . These entries are then grouped into K (possibly overlapping) partitions called *slots*. Each



Spatial Join. Figure 7. A seeded tree.



Spatial Join. Figure 8. Algorithms based on bulk-loading.



Spatial Join. Figure 9. Entries of an R-tree and a slot index built over them.

slot contains the MBR of the indexed R-tree entries, along with a list of pointers to these entries. Figure 9 illustrates a three-level R-tree (the leaf level is not shown) and a slot index built over it. If $K = 9$, the root level contains too few entries to be used as partition buckets. As the number of entries in the next level is over K , they are partitioned in nine (for this example) slots. The grouping policy used by SISJ is based on the R*-tree insertion algorithm. After building the slot index, all objects from the non-indexed relation are hashed into buckets with the same extents as the slots. If an object does not intersect any bucket it is filtered; if it intersects more than one buckets it is replicated. The join phase of SISJ loads all data from the R-tree under a slot and joins them (in memory) with the corresponding hash-bucket from the non-indexed dataset (in a similar way as HJ).

Comparison of Spatial Join Algorithms

Since indexes can facilitate the spatial join operation, algorithms (like RJ) that are based on existence of indexes are typically more efficient compared to methods that do not rely on indexes. For example, RJ (which

uses two R-trees) is expected to be more efficient than SISJ (which uses one R-tree), which is expected to be more efficient than HJ (which does not use trees).

RJ is the most popular index-based algorithm due to its efficiency and the fact that R-trees are becoming the standard access method in spatial database systems. Empirical and analytical studies have shown that the most efficient single-index methods are SISJ and SaM. Finally, conclusive results cannot be drawn about the relative performance of methods that do not consider indexes. S³J is expected to be faster than PBSM and HJ when the datasets contain relatively large rectangles and extensive replication occurs in HJ and PBSM. On the other hand, this method uses sorting which is more expensive than hashing, in general. SSSJ is also based on sorting, thus it could be more expensive than hash-based methods. Furthermore, sort-based methods do not favor pipelining and parallelism of spatial joins. On the other hand, the fact that PBSM uses partitions with fixed extents makes it suitable for processing multiple joins in parallel, since the space partitions (and the local joins for them) can be assigned to different processors.

Key Applications

Spatial Database Systems

The spatial join is a core operation of Spatial Database Management Systems [4].

Geographic Information Systems

A fundamental operator in GIS is map overlay. Given two thematically different maps of the same region (e.g., elevation and political), this operator produces a join map that emphasizes on the overlaps of objects from both joined maps. Spatial join is the database operator used to produce this output.

Data Mining

In many applications that handle high dimensional data, an important analysis operation is to discover groups of objects that are close to each other in the multidimensional space. Examples of such mining tasks include “find stocks with similar movements” (time-series databases) and “find pairs of similar images” (multimedia databases). This type of clustering of complex data objects can be performed with the help of spatial joins with distance predicates [7]. Simply speaking, the original objects (e.g., images) are approximated by high dimensional feature vectors, and a spatial self-join is applied on this space to derive pairs of nearby objects, which are then postprocessed to larger groups (clusters).

Cross-references

- ▶ [Hash Join](#)
- ▶ [Index Join](#)
- ▶ [Join](#)
- ▶ [Join Order](#)
- ▶ [Nested Loop Join](#)
- ▶ [Rtree](#)
- ▶ [Spatial Indexing Techniques](#)

Recommended Reading

1. Arge L., Procopiuc O., Ramaswamy S., Suel T., and Vitter J.S. Scalable sweeping-based spatial join. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 570–581.
2. Brinkhoff T., Kriegel H.-P., and Seeger B. Efficient processing of spatial joins using r-trees. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 237–246.
3. Corral A., Manolopoulos Y., Theodoridis Y., and Vassilakopoulos M. Closest pair queries in spatial databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 189–200.
4. Güting R.H. An introduction to spatial database systems. VLDB J., 3(4):357–399, 1994.

5. Guttman A. R-trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
6. Koudas N. and Sevcik K.C. Size separation spatial join. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 324–335.
7. Koudas N. and Sevcik K.C. High dimensional similarity joins: algorithms and performance evaluation. IEEE Trans. Knowl. Data Eng., 12(1):3–18, 2000.
8. Leutenegger S.T., Edgington J.M., and Lopez M.A. Str: a simple and efficient algorithm for R-tree packing. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 497–506.
9. Lo M.-L. and Ravishankar C.V. Spatial hash-joins. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 247–258.
10. Lo M.-L. and Ravishankar C.V. The design and implementation of seeded trees: An efficient method for spatial joins. IEEE Trans. Knowl. Data Eng., 10(1):136–152, 1998.
11. Mamoulis N. and Papadias D. Slot index spatial join. IEEE Trans. Knowl. Data Eng., 15(1):211–231, 2003.
12. Orenstein J.A. Spatial query processing in an object-oriented database system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1986, pp. 326–336.
13. Papadopoulos A., Rigaux P., and Scholl M. A performance evaluation of spatial join processing strategies. In Proc. 6th Int. Symp. Advances in Spatial Databases, 1999, pp. 286–307.
14. Patel J.M. and DeWitt D.J. Partition based spatial-merge join. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 259–270.
15. Preparata F.P. and Shamos M.I. Computational Geometry – An Introduction. Springer, 1985.

Spatial k-Anonymity

▶ [Spatial Anonymity](#)

Spatial Network Databases

BETSY GEORGE, SHASHI SHEKHAR
University of Minnesota, Minneapolis, MN, USA

Synonyms

[Spatial graph databases](#)

Definition

Spatial network databases render support for spatial networks by providing the necessary data model, query language, storage structure, and indexing methods. Spatial networks can be modeled as graphs where nodes are points embedded in space. One characteristic that distinguishes a spatial network database is the primary focus on the role of connectivity in relationships

rather than the spatial proximity between objects. These databases are the kernel of many important applications, including transportation planning; air traffic control; water, electric, and gas utilities; telephone networks; urban management; utility network maintenance, and irrigation canal management. The phenomena of interest for these applications are structured as a spatial graph, which consists of a finite collection of the points (i.e., nodes), the line-segments (i.e., edges) connecting the points, the location of the points and the attributes of the points and line-segments. For example, a spatial network database storing a road network may store road intersection points and the road segments connecting the intersections (Fig. 1).

Foundations

Data Model of Spatial Networks

This section presents techniques related to the data modeling of spatial networks. The database design involves three steps, namely conceptual modeling, logical modeling and physical modeling.

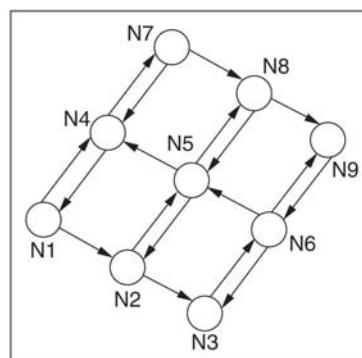
Conceptual Data Model: The purpose of conceptual modeling is to adequately represent the data types, their relationships and the associated constraints. The Entity Relationship (ER) model, widely used in conceptual modeling, does not offer adequate features to capture the spatial semantics of networks. The most critical feature of spatial networks, namely the connectivity between objects can be expressed, using a graph framework. At the conceptual level, the pictogram enhanced ER (PEER) model [13] can be used. Figure 2 shows a PEER diagram for a spatial network. In a

spatial graph, vertices represent road intersections and edges represent road segments. A path represents a street and consists of a series of edges.

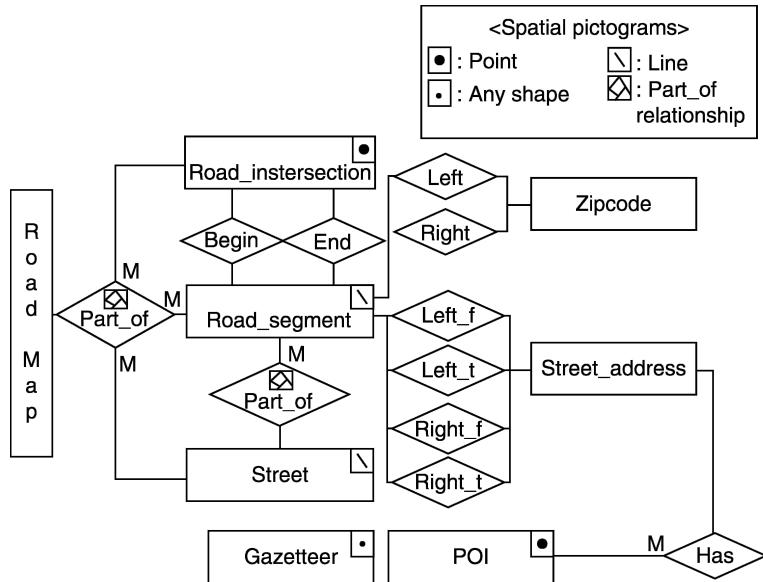
Labels and weights can be attached to vertices and edges to encode additional information such as names and travel times. Two edges are considered to be adjacent if they share a common vertex.

Modifications to the spatial network model have been proposed to make it more suitable in the context of some applications. For example, a simple node-edge network model might not be adequate to represent all features of a transportation network [9]. To address such limitations, various enhanced models have been proposed. One such model is the transportation data model (UNETRANS) that organizes the data model as three layers, namely (i) a reference network layer that represents the topological structure of the network, (ii) a route features layer that defines more complex features such as routes from the elements of the reference network layer, and (iii) the events layer that represents events such as traffic signs [2].

Logical Data Model: In the logical modeling phase, the conceptual data model is implemented using a commercial database management system. Among the various implementation models such as hierarchical, network, relational, object-relational data models and object-oriented models, the object-relational model has been gaining popularity in the representation of spatial applications. To model spatial network databases, graphs can be embedded into object-relational models. Shekhar and Chawla [11] lists some common graph operations used by spatial network applications,



Spatial Network Databases. Figure 1. A Road Map and its Spatial Network Representation. (a) A road map (b) Spatial Graph Representation (Source for Figure 1(a): <http://maps.yahoo.com>).



Spatial Network Databases. Figure 2. A PEER Diagram for Spatial Graph for a Road Network.

using a high-level object oriented notation that employs three fundamental classes in graphs, namely, Graph, Vertex, and Edge. Models such as GraphDB [5], which allow additional data types such as path, have also been proposed. A path class explicitly stores paths or routes in a graph, which contains the list of edges and nodes. In this model, an operator called “rewrite” can apply transformations to subsequences of heterogeneous sequences such as paths.

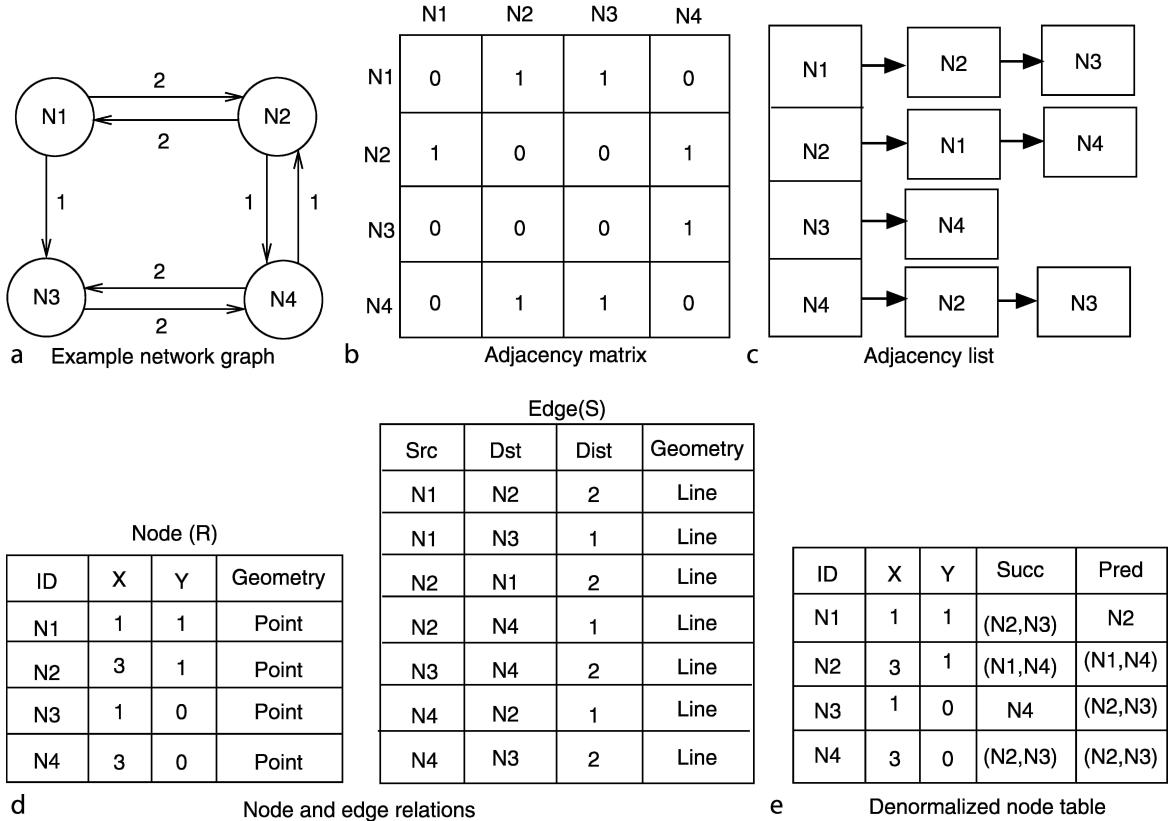
Physical Data Model: The physical data modeling phase deals with the actual implementation of the database application. Issues related to storage, indexing and memory management are addressed in this phase. Very often, queries that are posed on a network database such as a road map, involve route finding. This means the database must provide adequate support for network computations such as finding shortest paths.

Figure 3 shows three representations of a graph. Adjacency-matrix and adjacency list are two well-known data structures used for implementing road networks represented as graphs [11]. In an adjacency-matrix, the rows and columns of a matrix represent the vertices of the graph. A matrix entry can be either 1 or 0, depending on whether there is an edge between the two vertices as shown in **Fig. 3b**. An adjacency list (shown in **Fig. 3c**) consists of an array of pointers. Each element of the array represents a vertex in the graph and the pointer points to a list of vertices that are adjacent to the vertex. Directed graphs can be implemented in

the relational model using a pair of relations, one for the nodes and the other for the edges. The “Node” (R) and the “Edge” (S) relations are shown in **Fig. 3d** and a denormalized representation is shown in **Fig. 3e**. The denormalized representation of a node table contains the coordinates of the node, a list of its successors and a list of its predecessors. This representation is often used in shortest path computations.

A spatial access method called the Connectivity-Clustered Access Method (CCAM) was proposed in [12], which clusters the vertices of the graph based on graph partitions, thus providing an ordering based on connectivity.

Graph Algorithms: Frequent queries on a spatial network involve operations such as shortest path, nearest neighbor search, range search and closest pairs [10]. “Shortest” path algorithms find the least cost path between two nodes in a given graph. The cost of the path could be based on network distance, travel time or a user specified factor. Examples of popular shortest path algorithms are Dijkstra’s algorithm and A* search. Nearest neighbor search algorithms find the point(s) closest to a given query point. Traditionally, the closest point was determined based on the Euclidean distances, which did not consider the network connectivity of the objects. However, in practice, trajectories of objects are usually constrained by an underlying spatial network such as a road network and hence algorithms that find nearest neighbors and closest pairs that



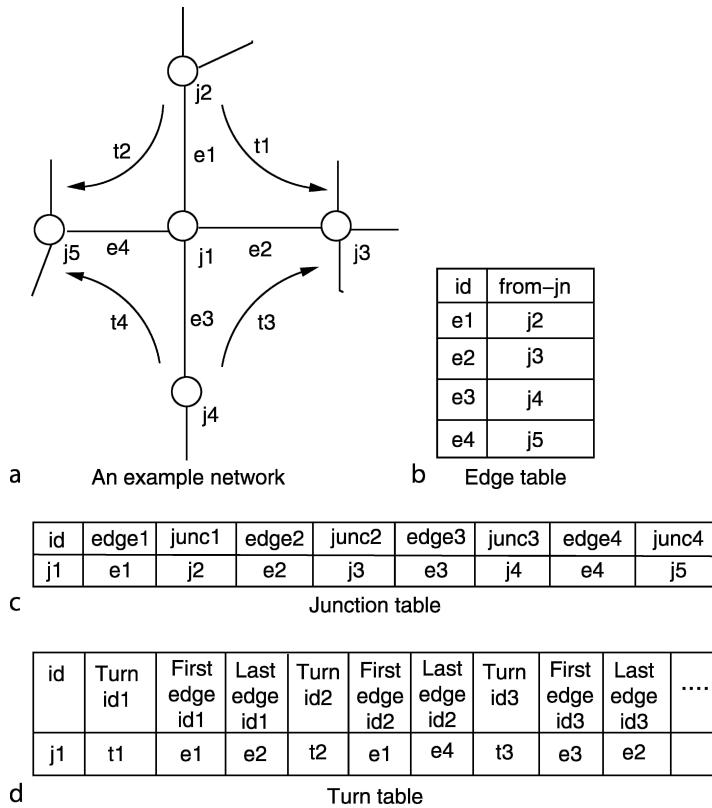
Spatial Network Databases. Figure 3. Three different representations of a graph.

consider network connectivity are critical in a spatial network database. Query processing algorithms that find nearest neighbors and closest pairs have been proposed [7,10].

Turn Restrictions: Turn restrictions are frequently encountered in road networks and they can affect the traversal in the network. A physical model that does not consider turn restrictions can lead to the computation of routes that are not entirely feasible. Turns have been modeled using a turn table where each turn restriction is represented as a row in the table that references the two associated edges [9]. Another proposed method to represent turn restrictions is node expansion [1]. The node that corresponds to a junction is expanded to a subgraph where permissible turns are represented as edges. This technique can lead to a substantial increase in the size of the network, which adversely affects the performance. Another method involves the transformation of the road network to a line graph where the edges in the original network are mapped to vertices in the line graph and the turns are represented as edges in the line graph [15].

A representation, consisting of a junction table, edge table and turn table was proposed in [6]. Every junction is represented as a row in the junction table. A row corresponding to a junction stores the edges that converge at the junction and the junctions connected to the given junction. The edge table stores edge identifiers and the junction where the edge originates (from-junction). A tuple in the turn table corresponds to a junction in the network. Each tuple consists of a junction identifier, and a triplet (turn identifier, first edge-id, last edge-id) corresponding to each turn associated with the given junction.

Figure 4 illustrates the representation of turn restrictions in a road network. Figure 4a shows a part of a road network around a junction $j1$ where the edges $e1, e2, e3$ and $e4$ meet. The curved arrows indicate the permitted turns at the junction. For example, a turn is allowed from edge $e1$ to edge $e2$. Figures 4b–4d show the edge, junction and turn tables respectively, corresponding to turn $t1$ in the example ($e4$) and the junctions connected to it ($j2, j3, j4$, and $j5$). The turn table shows the permitted turns at junction $j1$ and the edges



Spatial Network Databases. Figure 4. Representation of Turn Restrictions (adapted from [6]).

that participate in each turn. For example, turn t_1 represents a turn from edge e_1 to edge e_2 as illustrated by the “first edge id” and “last edge id” entries in the turn table in Fig. 4d.

Key Applications

Location-Based Services

Spatial network databases are indispensable for any location-based service that involves route based queries [14]. Location-based services (LBS) provide the ability to find the geographical location of a mobile device and subsequently provide services based on that location. Spatial network databases play a key role in providing efficient query-processing capabilities such as finding the nearest facility (e.g., a restaurant) and the shortest path to the destination from a given location. Route-finding queries typically deal with route choice (shortest route to a given destination), destination choice (the nearest facility from the given location) and departure time choices (the time to start the journey to a destination so that the travel time is

minimized). Though a significant amount of work has been done to find best routes and destinations, the problem of computing the best time to travel on a given route (time choice) needs further exploration.

Emergency Planning

One key step in emergency planning is to find routes in a road network to evacuate people from disaster-stricken areas to safe locations in the least possible time. This requires finding shortest routes from disaster areas to destinations. In metropolitan-sized transportation networks, manual computation of the required routes is almost impossible, making digital road maps integral to the efficient computation of these routes.

Future Directions

A significant fraction of queries that are posed on a road network involves finding the shortest path between a pair of locations. Travel times on the road segments very often depend on the time of day due to varying levels of congestion, thus making the

shortest paths also time-dependent. Road networks need to be modeled as spatio-temporal networks to account for this time-dependence. Various models such as time-expanded networks [8] and time-aggregated graphs [4,3] are being explored in this context. A time expanded graph represents the time-dependence by copying the network for every time instant whereas in time aggregated graphs, the time-varying attributes are aggregated over edges and nodes.

Cross-references

- ▶ [Graph](#)
- ▶ [Graph Database](#)
- ▶ [Road Networks](#)

Recommended Reading

1. Anez J., de la Barra T., and Perez B. Dual graph representation of transport networks. *Transport. Res.*, 30(3):209–216, 1996.
2. Curtin K., Noronha V., Goodchild M., and Grise S. ARCGIS Transportation Model (UNETRANS), UNETRANS Data Model Reference, December 2003.
3. George B. and Shekhar S. Time-aggregated graphs for modeling spatio-temporal networks – an extended abstract. In Proc. Workshops at Int. Conf. on Conceptual Modeling, 2006, pp. 85–99.
4. George B. and Shekhar S. Spatio-temporal network databases and routing algorithms: a summary of results. In Proc. 10th Int. Symp., Advances in Spatial and Temporal Databases, 2007, pp. 460–477.
5. Guting R.H. GraphDB: modeling and querying graphs in databases. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994.
6. Hoel E.G., Heng W.L., and Honeycutt D. High performance multimodal networks. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005.
7. Jensen C.S., Kolar J., Pederson T.B., and Timko I. Nearest neighbor queries in road networks. In Proc. 11th ACM Int. Symp. on Advances in Geographic Inf. Syst., 2003.
8. Kohler E., Langtau K., and Skutella M. Time-expanded graphs for flow-dependent transit times. In Proc. Tenth Annual European Symp. on Algorithms, 2002.
9. Miller H.J. and Shaw S.L. GIS-T Data Models, Geographic Information Systems for Transportation: Principles and Applications. Oxford University Press, Oxford, 2001.
10. Papadias D., Zhang J., Mamoulis N., and Tao Y. Query processing in spatial network databases. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003.
11. Shekhar S. and Chawla S. Spatial Databases: A Tour. Prentice Hall, Englewood Cliffs, NJ, 2002.
12. Shekhar S. and Liu D.R. CCAM: a connectivity-clustered access method for networks and network computations. *IEEE Trans. Knowl. Data Eng.*, 9(1):102–119, 1997.
13. Shekhar S., Vatsavai R., Chawla S., and Burke T.E. Spatial pictogram enhanced conceptual data models and their translation to logical data models. In Proc. Int. Workshop on Integrated spatial databases, digital maps, and GIS, 1999.

14. Shekhar S., Vatsavai R., Ma X., and Yoo J. Navigation systems: a spatial database perspective, Chapter 3. In *Location-Based Services*, J. Schiller, A. Voisard (eds.). Morgan Kaufmann, 2004.
15. Winter S. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002.

Spatial Operations and Map Operations

MICHEL SCHOLL¹, AGNÈS VOISARD^{2,3}

¹Cedric-CNAM, Paris, France

²Fraunhofer Institute for Software and Systems Engineering (ISST), Berlin, Germany

³Free University of Berlin, Berlin, Germany

Synonyms

[Map Algebra](#); [Layer Algebra](#); [Theme Algebra](#)

Definition

Map operations refer to the operations that an end user performs on maps stored in a database. Map information is stored according to themes (for instance, cities, roads, or population), sometimes called layers in the GIS terminology. The maps considered here are stored in a vector format – as opposed to a raster format such as a grid of pixels – and can be 1 dimensional (e.g., a network of roads for a navigation system), 2 dimensional (e.g., a map of land-use for regional planning activities), or 2.5 dimensional if the elevation at certain locations is considered (for instance, the height of a building in an architecture project). A map is made of what is often called *geographic objects*. A geographic object (for instance, a city) has two parts, an alphanumeric one (e.g., its name and population) and a spatial one (e.g., a polygon), usually called *spatial object*. The alphanumeric attributes of a geographic object constitute its description. Map operations may use operations on spatial objects, commonly referred to as *spatial operations*. When describing the structure of a map – its description and its spatial part – together with its associated operations, one refers to a *map model*. The same applies to the structure and behavior of the spatial part of the geographic objects, leading to a *spatial model*.

Historical Background

When data are stored in a database, it is accessed through a query language, such as SQL. In the case of

maps, however, SQL needs to be extended in order to consider their spatial component. With the emergence of GIS in the 80s, and also because maps are particular “non standard” entities, a set of operations to manipulate them was proposed by database researchers to describe these operations at a high level of abstraction, i.e., without considering SQL details. The idea was to define, at a conceptual level, unary or binary operations on maps that possibly take other arguments (alphanumeric or spatial) and that return maps, hence the term *algebra* often used in this context. Such operations also need spatial operations (geometric or topological) such as the intersection of polygons. Many proposals for lists of spatial operations were made. [8] is one of the first attempts to describe general map operations from a GIS view point. [7] proposed an extensible query language to the designer of geographic databases, independent of any underlying database model. The geo-relational algebra [2] was a pioneer approach, proposing an algebra based on the relational model that encompassed spatial operations. The SpatialSQL language [1] includes a list of spatial operations to be eventually used in conjunction with SQL. Operations on thematic layers were also proposed (e.g., [4]). The ROSE algebra [3] is a rich approach based on the relational model that allows extensible sets of functions. The OGIS Standard for SQL [5] from the Open GIS Consortium (OGC) focuses on spatial operations to be integrated in SQL and proposes an exhaustive list of such operations. Most current commercial approaches such as Oracle or ArcGIS from ESRI offer data types and operations that are OGIS compliant.

Foundations

This part focuses on end-user map operations that are performed in a database. In current applications, maps are usually stored in a relational database extended to abstract spatial data types. A kernel of elementary operations that can be combined in order to answer complex queries is presented here. The following list is coming from [2,6].

Map model. A map M is defined as a set of geographic objects: $M = \{g\}$ where a geographic object g is defined as follows: $g = \{<A>, S\}$, where $<A>$ is a list of alphanumeric attributes and S a spatial attribute.

Spatial model. The spatial attribute S of a geographic object corresponds to its associated geometric part (it also has topological relationships with other

objects). It can be simple (for instance, one polygon for a lake) or complex, i.e., made of many parts (for instance, many polygons for a given country and its islands). A spatial attribute has a certain type and can be 0-, 1-, or 2-dimensional. Note that dimensions are often not mixed in a spatial attribute. An entity with a spatial type is usually called a spatial object and the referential used is the Euclidean plane.

The basic types of spatial objects that are usually considered are:

POINT	a 0-dimensional spatial object
LINE	a 1-dimensional spatial object made of segments
REG	a 2-dimensional spatial object made of polygons

as well as sets of these objects.

Spatial Operations

The operations presented below are primitives on spatial objects that are used in map operations. In the following, their signature is used for their short description (i.e., the type of arguments that these operations take and the one that they return). The spatial operations are presented here according to four groups: spatial predicates, spatial extractions, set operations, and geometric operations. Other classifications based for instance on the types of arguments are also sensible. Note also that the list given below is not exhaustive.

Map Operations

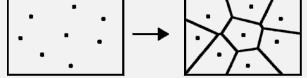
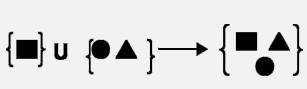
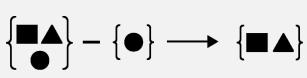
The operations described below constitute a common set of operations on maps. They are illustrated using the map of the 12 districts of Berlin, Germany, or a subset of it in some cases, and using the following schemas:

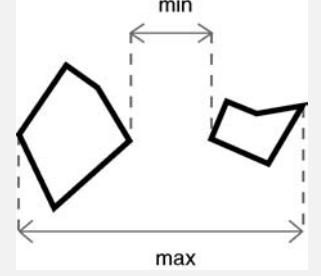
District (Name:STRING, Population:NUM, Area:REG)

CityDivision (Name:STRING, Area:REG)

Other Operations

The list of operations given above is not exhaustive but it corresponds to a kernel of common general operations on thematic maps. They are typical database operations performed in a GIS. Other GIS operations that are not detailed here include classification, zoom in, zoom out, as well as operations on layers stored in a raster form.

Group 1: Spatial Predicates		
In the following, BOOL represents a Boolean value (true or false) and REG is an abbreviation for the REGION type.		
Different 	Possible signatures:	Tests whether two spatial objects are different in the plane $POINT \times POINT \rightarrow BOOL, LINE \times LINE \rightarrow BOOL, REG \times REG \rightarrow BOOL$
Equal 	Possible signatures:	Tests whether two spatial objects are the same (i.e., have the same value in the plane) $POINT \times POINT \rightarrow BOOL, LINE \times LINE \rightarrow BOOL, REG \times REG \rightarrow BOOL$
Intersects 	Possible signatures:	Tests whether two spatial objects intersect $LINE \times LINE \rightarrow BOOL, LINE \times REG \rightarrow BOOL, REG \times REG \rightarrow BOOL$
Inside/Outside 	Possible signatures:	Tests whether a spatial object is inside/outside a given region $POINT \times REG \rightarrow BOOL, LINE \times REG \rightarrow BOOL, REG \times REG \rightarrow BOOL$
Adjacent 	Possible signatures:	Tests whether two spatial objects are adjacent (i.e., have a common boundary) $LINE \times REG \rightarrow BOOL, REG \times REG \rightarrow BOOL$
Group 2: Spatial Extractions		
These operators transform their spatial input into another type of spatial object.		
Intersection 	Possible signatures:	Returns the intersection of two spatial objects $LINE \times LINE \rightarrow \{POINT\}, LINE \times LINE \rightarrow LINE, LINE \times REG \rightarrow \{POINT\}, LINE \times REG \rightarrow LINE, REG \times REG \rightarrow \{POINT\}, REG \times REG \rightarrow LINE, REG \times REG \rightarrow REG$
Voronoi 	Possible signatures:	Returns the Voronoi diagram of a region. Note that the returned set of regions has the particularity that the regions do not overlap $\{POINT\} \times REG \rightarrow \{REG\}$
Closest 	Possible signatures:	Returns the closest spatial object from a given object, taken from a set $POINT \times \{POINT\} \rightarrow POINT, POINT \times \{LINE\} \rightarrow LINE, POINT \times \{REG\} \rightarrow REG, LINE \times \{POINT\} \rightarrow POINT, LINE \times \{LINE\} \rightarrow LINE, LINE \times \{REG\} \rightarrow REG, REG \times \{POINT\} \rightarrow POINT, REG \times \{LINE\} \rightarrow LINE, REG \times \{REG\} \rightarrow REG$
Group 3: Set Operations		
The following operations are common operations on sets of objects. Note again that a set only contains entities of the same type.		
SetUnion 	Possible signatures:	Returns the union of two sets of spatial objects, in the mathematical sense $\{POINT\} \times \{POINT\} \rightarrow \{POINT\}, \{LINE\} \times \{LINE\} \rightarrow \{LINE\}, \{REG\} \times \{REG\} \rightarrow \{REG\}$
SetDifference 	Possible signatures:	Returns the difference of two sets of spatial objects $\{POINT\} \times \{POINT\} \rightarrow \{POINT\}, \{LINE\} \times \{LINE\} \rightarrow \{LINE\}, \{REG\} \times \{REG\} \rightarrow \{REG\}$
SetIntersection 	Possible signatures:	Returns the intersection of two sets of spatial objects $\{POINT\} \times \{POINT\} \rightarrow \{POINT\}, \{LINE\} \times \{LINE\} \rightarrow \{LINE\}, \{REG\} \times \{REG\} \rightarrow \{REG\}$

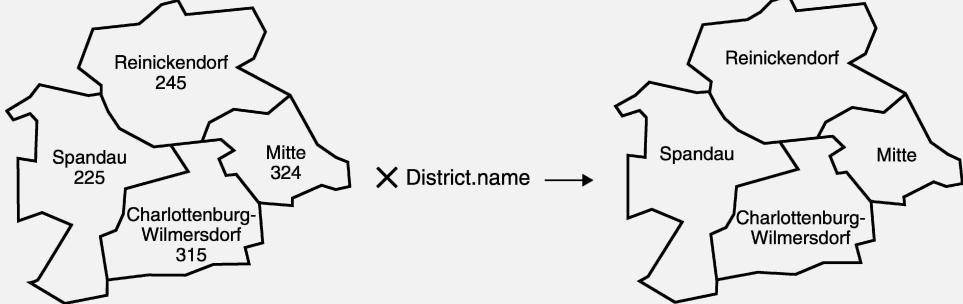
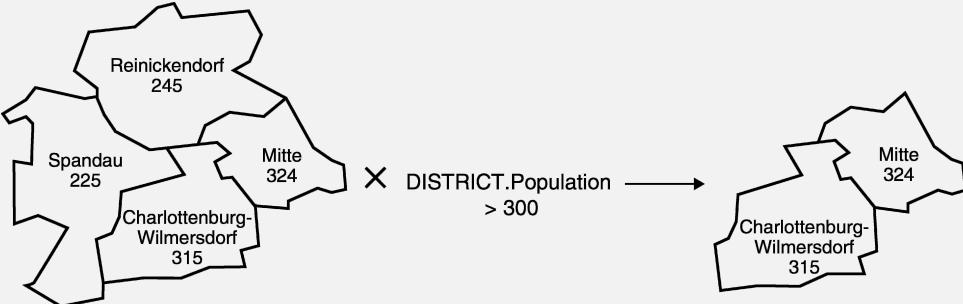
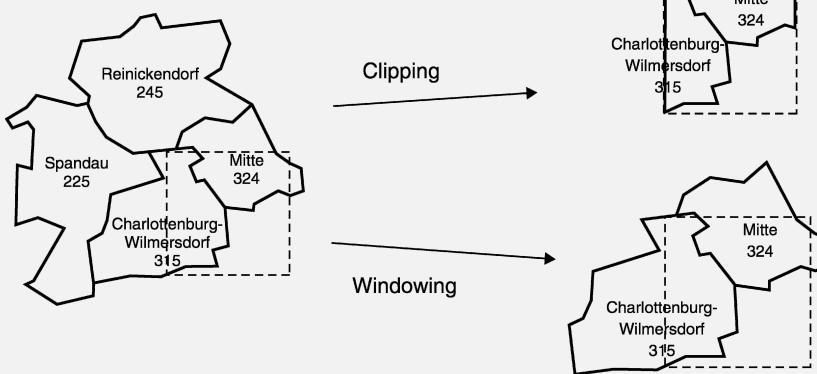
Group 4: Geometric Operations		
In the following, NUM represents a numerical value.		
Convex Hull 	Returns the region (polygon) that encompasses all the points given in the argument set	Signature: $\{POINT\} \rightarrow REG$
Center 	Returns the center of a set of points	Signature: $\{POINT\} \rightarrow POINT$
Min/Max Distance 	Returns the minimal (respect. maximal) distance between two spatial objects	Possible signatures: $POINT \times LINE \rightarrow NUM, LINE \times LINE \rightarrow NUM, POINT \times REG \rightarrow NUM, LINE \times REG \rightarrow NUM, REG \times REG \rightarrow NUM$
Length 	Returns the length of a line	Signature: $LINE \rightarrow NUM$
Perimeter 	Returns the perimeter of a region. In case the region is composed of many polygons, it returns the sum of their respective perimeters	Signature: $REG \rightarrow NUM$
Area 	Returns the area of a region (total area if it is composed of many polygons)	Signature: $REG \rightarrow NUM$

Key Applications

Users likely to eventually use these map operations are end-users who need to get information from existing maps, for instance for the purpose of planning or geo-marketing. However, this conceptual approach – which moves away from implementation details – is targeted towards spatial database application designers who need to design appropriate application environments. Such environments should be easy to use,

extensible, and adaptable to various application needs. They should, moreover, offer an efficient operation processing, however, this aspect is not handled by the conceptual approach presented here.

The key applications of this area concern thematic map manipulation (by the census bureau, city planners, local administrators, transportation managers, and so on) in order to perform statistics and analysis on data having a spatial dimension.

Map Projection	Returns a map having a list of attributes given as argument and an unchanged spatial part
	Signature: $map \times <A> \rightarrow map$, where $<A>$ is a collection of alphanumerical attributes
	
Map Selection	Returns a map whose geographic objects satisfy the selection criteria given as argument (e.g., population greater than 300 thousand inhabitants)
	Signature: $map \times \text{selection-criteria } (<A>) \rightarrow map$, where selection-criteria ($<A>$) is a predicate on one or many alphanumerical attributes
	
Spatial Selections	- Windowing (or region query): Returns the map made of the original map whose objects intersect the region given as argument (often, a rectangle)
	- Clipping: Returns the part of the map that is exactly in the region given as argument
	Signature: $map \times \text{REG} \rightarrow map$
	

Map Overlay	Generates a new map from two (overlaid) maps, for instance a map of the former city division in Berlin and the map of districts. It uses the intersection operation on spatial objects. It creates new geographic objects as can be seen in the center of the newly created map. This operation is also called a spatial join in the database terminology	
	Signature:	$map \times map \rightarrow map$
	Note:	For legibility reasons, the following map of districts does not include their population.
Map Union	Returns a map made of the two arguments	
	Signature:	$map \times map \rightarrow map$
Fusion	Performs the geometric union of the spatial part of geographic objects that belong to the same map. Note the sum of the population in the example	
	Signature:	$map \rightarrow map$

Cross-references

- ▶ [Geographic Information System](#)
- ▶ [OGC](#)
- ▶ [Semantic Modeling for Geographic Information Systems](#)
- ▶ [Spatial Data Types](#)

Recommended Reading

1. Egenhofer M.J. Spatial SQL: a query and presentation language. *IEEE Trans. Knowl. Data Eng.*, 6(1):86–95, 1994.
2. Güting R.H. Geo-relational algebra: a model and query language for geometric database systems. In *Advances in Database Technology*, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp. 506–527.
3. Güting R.H. and Schneider M. Realm-based spatial data types: the ROSE algebra. *VLDB J.*, 4(2):243–286, 1995.
4. Hadzilacos T. and Tryfona N. Logical data modelling for geographical applications. *Intl. J. Geogr. Inf. Sci.*, 10(2):179–203, 1996.
5. Open GIS Consortium. OpenGIS® Geographic objects implementation specification, 2007.

6. Rigaux P., Scholl M., and Voisard A. Spatial Databases – With Application to GIS, Chapter 3. Morgan Kaufmann/Elsevier, 2001.
7. Scholl M. and Voisard A. Thematic map modeling. In Proc. Int. Symp. on Spatial Databases, 1989, pp. 167–190.
8. Tomlin D. A Map algebra. In Proc. Harvard Computer Graphic Conf., 1983.

Spatial Outliers

- ▶ [Spatial Data Mining](#)

Spatial Referencing

- ▶ [Georeferencing](#)

Spatial Statistics

- ▶ [Spatial Data Mining](#)

Spatio-Temporal Approximation

- ▶ [Spatiotemporal Interpolation Algorithms](#)

Spatio-Temporal Benchmarking

- ▶ [Real and Synthetic Test Datasets](#)

Spatio-Temporal Data Generator

- ▶ [Real and Synthetic Test Datasets](#)

Spatio-Temporal Data Mining

NIKOS MAMOULIS

University of Hong Kong, Hong Kong, China

Synonyms

[Data mining in moving objects databases](#)

Definition

The extraction of implicit, non-trivial, and potentially useful abstract information from large collections of spatio-temporal data are referred to as spatio-temporal data mining. There are two classes of spatio-temporal databases. The first category includes timestamped sequences of measurements generated by sensors distributed in a map, and temporal evolutions of thematic maps (e.g., weather maps). The second class are moving object databases that consist of object trajectories (e.g., movements of cars in a city). A trajectory can be modeled as a sequence of (p_i, t_i) pairs, where p_i corresponds to a spatial location and t_i is a timestamp. The management and analysis of spatio-temporal data has gained interest recently, mainly due to the rapid advancements in telecommunications (e.g., GPS, Cellular networks, etc.), which facilitate the collection of large datasets of object locations (e.g., cars, mobile phone users) and measurement sequences (e.g., sensor readings). Mining tasks for moving object databases include detection and prediction of traffic jams, analyzing the movement behavior of animals, clustering or classification of moving objects according to their direction and/or speed, and identification of trends that associate the movement/speed of objects to their destination. In addition, from databases of measurement sequences, spatial relationships between correlated or anticorrelated sequences can be extracted (e.g., “sensors within 10 m from each other produce similar readings with high probability”), or build classification models to detect abnormal combinations of sensor readings. The analysis of spatio-temporal databases is challenging due to the vast amount of collected data, and the complexity of novel mining tasks. Special issues include the fuzzy and implicit nature of spatio-temporal relationships between objects, the complex geometry of spatial objects, the varying temporal nature of events (instantaneous vs. durable), the variability of spatio-temporal data (moving objects, evolution of spatial events or phenomena, etc.), and the multiple (spatial and temporal) resolution levels of abstraction.

Historical Background

Data mining became a core field of database research in the 1990s [6]. Initial research focused on mining tasks (association analysis, classification, and clustering) applied on relational databases, or transactional, data that record sets of items purchased together. Two parallel streams of research were born soon after the

first papers; data mining for temporal and spatial data. Temporal data mining focused on the extraction of sequential patterns from ordered transactional data or event sequences. Clustering and classification of time series, a classic problem in statistics, also triggered the interest of this research stream. The spatial relationships between associated events has been the main focus of spatial data mining. Spatio-temporal data mining is a rather new research field. Initially [4,11], temporal data mining techniques were applied for spatio-temporal data, after modeling the input as multi-dimensional temporal sequences. Lately, new problems, particular to this type of data have emerged, such as clustering multidimensional trajectories [12], clustering or pattern mining based on common subtrajectories [2,8] mining periodic patterns in moving object trajectories [9] and discovery of moving clusters with partial membership of objects during a cluster's lifetime [7].

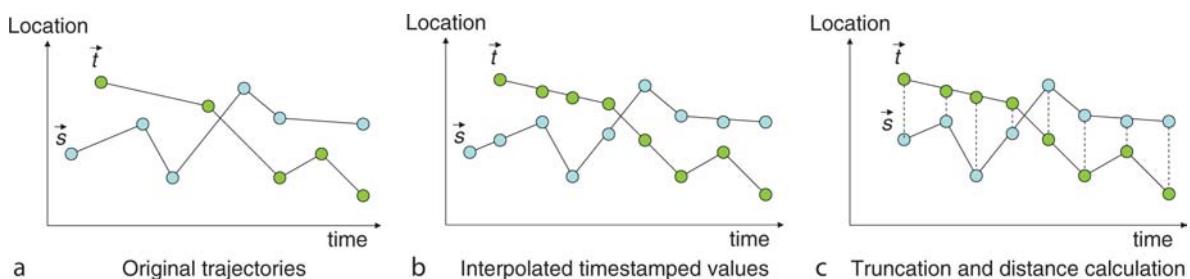
Foundations

Clustering

Clustering is a classic data mining task that divides a set of objects into groups (clusters), such that the objects in the same cluster are similar to each other and objects in different clusters are dissimilar. Most clustering prototypes [6] can be applied for spatio-temporal data after (i) the objects to be clustered are well-defined, and (ii) a distance function between objects has been determined. The first prototype is partitioning-based clustering, where a set of k initial random partitions are iteratively refined. Characteristic algorithms in this class are the k -means and k -medoids algorithms. Hierarchical methods is another category of algorithms, where initially each object forms a cluster on its own and clusters are merged iteratively until a convergence criterion is met. Another

popular clustering prototype is density-based clustering, where dense regions of nearby objects are iteratively merged. These general methods will not be discussed in detail here; instead, the focus will be on similarity measures for spatio-temporal data and on special definitions of clustering in this context.

Clustering can be performed on trajectories in order to classify moving objects into groups of similar movement behavior. The distance between two trajectories can be defined after modeling them as high-dimensional vectors, by some preprocessing if necessary. More specifically, if the timestamps of trajectories do not match, any missing timestamped values are generated by interpolation, such that the transformed trajectories correspond to vectors of the same length and they have the same timestamps (as regular as possible). In addition, if the sequences have different timespan they are shrunk (or extended) as necessary by truncation (or interpolation). Eventually, the distance between the resulting trajectories is measured using some appropriate distance function (e.g., Euclidean distance). Figure 1 illustrates this process. Apart from the Euclidean distance, other measures specific to (multidimensional) time series have been proposed. A popular distance measure is dynamic time warping (DTW) [1]. DTW allows elastic shifting of sequence in order to detect similar shapes with different phases. DTW aligns each element of one sequence to one or more elements of the other sequence by applying a dynamic programming process, similar to the edit distance computation between strings. Another method to assess the similarity between two time series is to find their longest common subsequence (LCSS) [3]. One definition of the LCSS between two vectors \vec{s} and \vec{t} is the pair of subsequences $\vec{s}' \in \vec{s}$ and $\vec{t}' \in \vec{t}$, such that (i) \vec{s}' and \vec{t}' have the same length, (ii) $dist(\vec{s}', \vec{t}') \leq \epsilon$, and (iii) (\vec{s}', \vec{t}') are the longest subsequences that qualify the distance constraint

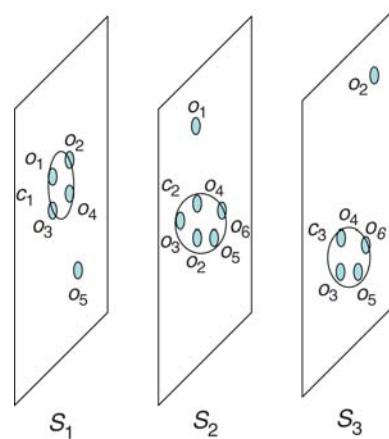


Spatio-Temporal Data Mining. Figure 1. Measuring distance between trajectories.

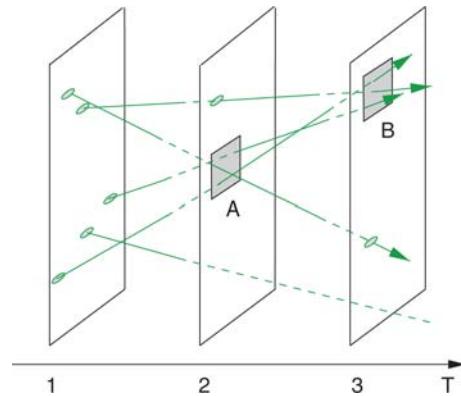
expressed in (ii). Here, $dist()$ is a basis measure between sequences (e.g., Euclidean distance) and ε is a threshold quantizing adequate closeness. LCSS (like DTW) allows stretching of sequences in time (provided that a DTW-like measure is used as measure), and at the same time is more robust to noise, giving weight to the similar portions of the sequences and ignoring the very different parts. In [12] an appropriate definition of LCSS is given for multidimensional time series (i.e., trajectories), where a constraint for the maximum shift in time between common subsequences exists, and different stretching/shifting at different dimensions can also be performed in order to reach the best possible matching.

Instead of applying classic clustering algorithms with the help of an appropriate distance measure, [4] define clusters of trajectories by mixtures of regression models, which are extracted with the help of an EM algorithm. In a recent work, [8] clusters trajectories, after partitioning them to line segments. The clusters in this case are formed by similar line segments, and a trajectory may belong to more than one cluster.

A special definition of moving clusters is presented in [7]. The difference compared to the trajectory clusters defined above is that the identity of a moving cluster remains unchanged, while its location and content may change over time. For example, while a group of animals are migrating, some new animals may enter the group (e.g., those passing nearby the clusters trajectory or newborns), while some animals may leave the group (e.g., those attacked and eaten by lions). Formally, consider a set of moving objects in a long, timestamped history $H = \{t_1, t_2, \dots, t_n\}$. A *snapshot* S_i of H is the set of objects and their locations at time t_i . Given a snapshot S_i , a standard spatial clustering algorithm can be employed to identify dense groups of objects in S_i which are close to each other. Let c_i and c_{i+1} be two such *snapshot clusters* for S_i and S_{i+1} , respectively. Then $c_i c_{i+1}$ is said to be a *moving cluster* if $\frac{|c_i \cap c_{i+1}|}{|c_i \cup c_{i+1}|} \geq \theta$, where θ ($0 < \theta \leq 1$) is an integrity threshold for the contents of the two clusters. Intuitively, if two spatial clusters at two consecutive snapshots have a large percentage of common objects, then these are considered as a single cluster that moved between these two timestamps. Figure 2 shows an example of a moving cluster. S_1 , S_2 , and S_3 are three snapshots. In each of them there is a timeslice cluster (c_1 , c_2 , and c_3). Let $\theta = 0.5$. $c_1 c_2 c_3$ is a moving cluster, since $\frac{|c_1 \cap c_2|}{|c_1 \cup c_2|} = \frac{3}{6} = 0.5$ and $\frac{|c_2 \cap c_3|}{|c_2 \cup c_3|} = \frac{4}{5} = 0.8$ are both



Spatio-Temporal Data Mining. Figure 2. Example of a moving cluster.



Spatio-Temporal Data Mining. Figure 3. Identifying areas of high object density.

at least θ . Note that objects may enter or leave the moving cluster during its lifetime. Using this definition, Kalnis et al. [7] extended a density-based clustering algorithm into methods that discover moving clusters in a large database of moving object trajectories.

Hadjieleftheriou et al. [5] proposed a framework for the discovery of areas with a high density of moving objects in the future, given the current locations and movements of a set of points, e.g., “Find all regions that will contain more than 500 objects, 10 min from now.” The time interval during which the areas remain dense is also part of the mining task. Figure 3 (taken from [5]) shows a graphical example of such dense areas (A in timestamp 2 and B in timestamp 3), assuming that at least three objects must exist in an area of one square unit for the area to be considered dense.

The discovery of dense regions is done after defining a space-time 3D grid and merging neighboring dense cells in this grid. Note that this problem is different to classic clustering, where objects are grouped based on the whole history (or trend) of their movement.

Classification and Prediction

Classification of trajectories is usually performed by nearest neighbor (NN) classifiers [12]. Given a trajectory \vec{s} of unknown label and a database D of labeled samples, such a classifier (i) searches in D for the k most similar time series to \vec{s} and (ii) gives \vec{s} the most popular label in the set of k returned time series. NN classifiers, like clustering algorithms, rely on an appropriate similarity function between trajectories. Depending on the application, one of the distance functions used for clustering (as discussed above) can be used.

A related task to classification is predicting the future movement of an object given its past locations. Regression models for one-dimensional time series can be extended for (multidimensional) moving object trajectories. The movement of objects is approximated by (mixtures of) functions, which in turn are used for prediction. Given the recent past movement of an object [10] propose a methodology that computes a recursive motion function; a concise form that captures a large number of movement types (e.g., polynomials, ellipses, sinusoids, etc.). A recursive function differs from classic regression functions in that it relates an object's location to those of the recent past.

Pattern Extraction

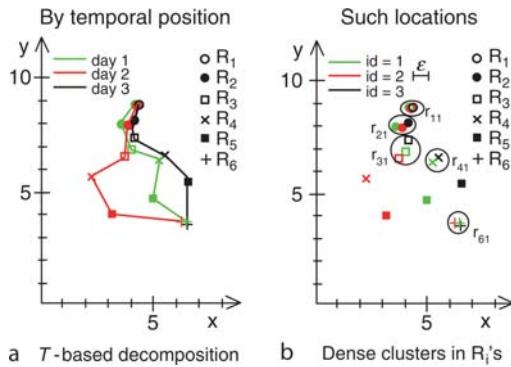
There is limited work on extraction of patterns from spatio-temporal databases, which has been treated as a generalization of pattern mining in time series data. For example, [11] studied the discovery of frequent patterns related to changes of natural phenomena (e.g., temperature changes) in spatial regions. The locations of objects or the changes of natural phenomena over time are converted to categorical values. For instance, the map can be divided into spatial regions and replace the location of the object at each timestamp, by the region-id where it is located. Similarly, the change of temperature in a spatial region can be modeled as a sequence of temperature values. Continuous domains of the resulting time series data are discretized, prior to mining. In the case of multiple

moving objects (or time series), trajectories are typically concatenated to a single long sequence. Then, an algorithm that discovers frequent subsequences in a long sequence (e.g., [13]) is applied.

In many applications, the movements obey periodic patterns; i.e., the objects follow the same routes (approximately) over regular time intervals. Objects that follow approximate periodic patterns include transportation vehicles (buses, boats, airplanes, trains, etc.), animal movements, mobile phone users, etc. For example, Bob wakes up at the same time and then follows, more or less, the same route to his work everyday. Periodic patterns can be thought of as (possibly non-contiguous) sequences of object locations that reappear in the movement history periodically.

Formally, let S be a sequence of n spatial locations $\{l_0, l_1, \dots, l_{n-1}\}$, representing the movement of an object (e.g., Bob) over a long history. Let $T \ll n$ be an integer called *period* (e.g., day, week, month). A *periodic segment* s is defined by a subsequence $l_{il+1} \dots l_{i+T-1}$ of S , such that $i \bmod T = 0$. Thus, segments start at positions $0, T, \dots, (\lfloor \frac{n}{T} \rfloor - 1) \cdot T$, and there are exactly $m = \lfloor \frac{n}{T} \rfloor$ periodic segments in S . A *periodic pattern* P is defined by a sequence $r_0 r_1 \dots r_{T-1}$ of length T , such that r_i is either a spatial region or $*$. The *length* of a periodic pattern P is the number of non-* regions in P . A segment s^i is said to *comply with* P , if for each $r_i \in P$, $r_i = *$ or s^i is *inside* region r_i . The *support* of a pattern P in S is defined by the number of periodic segments in S that comply with P . The same symbol P is used to refer to a pattern and the set of segments that comply with it. Let $\text{min_sup} \leq m$ be a positive integer (*minimum support*). A pattern P is *frequent*, if its support is larger than min_sup . Patterns for which the regions r_i are too sparse are not interesting, therefore a constraint is imposed to the density of these regions. Let S^P be the set of segments that comply with a pattern P . Then each region r_i of P is *valid* if the set of locations $R_i^P := \{s^i | s^i \in S^P\}$ form a *dense cluster*.

The discovery of partial periodic patterns from a long trajectory has been studied in [9]. This process is performed in two phases. First, S is divided into T spatial datasets, *one for each offset* of the period T . Specifically, locations $\{l_b, l_{b+T}, \dots, l_{b+(m-1) \cdot T}\}$ go to set R_b , for each $0 \leq b < T$ (m is the length of S). A spatial clustering algorithm is applied to discover clusters in each R_b . These clusters define periodic patterns of length 1. Figure 4 shows the spatial datasets obtained after

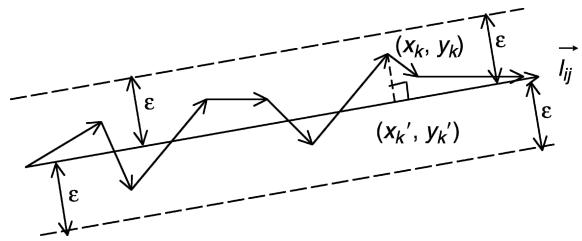


Spatio-Temporal Data Mining. Figure 4. Locations and regions per periodic offset.

decomposing the trajectory of an object in three consecutive days (periods). A different symbol is used to denote locations that correspond to different periodic offsets and different colors are used for different segment-ids. Observe that a dense cluster r in dataset R_i corresponds to a frequent pattern, having * at all positions and r at position i . Figure 4b shows examples of five clusters discovered in datasets R_1, R_2, R_3, R_4 , and R_6 . These correspond to five 1-patterns (i.e., $r_{11}****$, $*r_{21}****$, etc.).

In the second phase, [9] extend the Apriori algorithm [6] to identify longer patterns level-by-level. In specific, Pairs $\langle P_1, P_2 \rangle$ of frequent $(k - 1)$ -patterns with their first $k - 2$ non-* regions in the same position and different $(k - 1)$ -th non-* position create candidate k -patterns, the supports of which are counted at the next pass of the data sequence.

There has also been research on spatio-temporal pattern mining, where trajectories are regarded as sequences of locations (without giving any importance to the timestamps). In this case, the objective is to extract route-patterns of moving objects irrespectively to their speed. In this spirit, [2] define spatio-temporal patterns as sequences of line segments that form frequently followed routes by moving objects. A line simplification algorithm is used to approximate subsequences of trajectories by line segments. Figure 5 illustrates a subsequence s_{ij} which is approximated by a line segment \vec{l}_{ij} , such that the maximum distance of any point from s_{ij} to its projection on \vec{l}_{ij} is at most ε . Simplified line segments of subsequences are clustered to form spatial regions (pattern elements) that can approximate a large number of subsequences. If a region approximates more than min_sup subsequences, then



Spatio-Temporal Data Mining. Figure 5. Subsequence approximation.

it forms a frequent movement pattern of length 1. A generalized frequent movement pattern is an m -length ordered sequence of pattern elements that is supported by (i.e., approximates) more than min_sup subsequences. The enumeration of frequent patterns is performed in two phases; first the frequent 1-patterns are identified with the help of the clustering algorithm that is based on line simplification; then, longer patterns are found by employing a substring tree which compresses overlapping sequences of pattern elements.

Key Applications

Traffic Analysis

A motivating application of spatio-temporal data mining is to predict and analyze the causalities of traffic phenomena. Clustering or pattern extraction can help towards this purpose, since common routes that pass through the same map locations are likely the cause of traffic. Analyzing the causes of such clusters can lead to better transportation design for a city map.

Studying the Movement Behavior of Animals

With the help of GPS technology, the movements of animals can be tracked and analyzed. Identifying clusters or movement patterns can help in understanding the behavior of animals, such as the formulation and maintenance of herds, motion trends based on weather conditions, etc.

Video Analysis

The identification of objects and their movement behavior in video scenes is also an important application of spatio-temporal data mining. In this case, patterns of movement behavior can be extracted and analyzed.

For example, from a soccer game, one can extract the movement style of players. Or, from a martial arts video, one can analyze movement sequences of body parts and relate them to the objective of the subject.

Cross-references

- ▶ [Data Mining](#)
- ▶ [Geometric Stream Mining](#)
- ▶ [Spatial and Spatio-Temporal Data Models and Languages](#)
- ▶ [Spatial Data Mining](#)
- ▶ [Spatio-Temporal Data Warehouses](#)
- ▶ [Spatio-Temporal Trajectories](#)
- ▶ [Temporal Data Mining](#)

Recommended Reading

1. Berndt D. and Clifford J. Using dynamic time warping to find patterns in time series. In Proc. KDD Workshop, 1994.
2. Cao H., Mamoulis N., and Cheung D.W. Mining frequent spatio-temporal sequential patterns. In Proc. 2005 IEEE Int. Conf. on Data Mining, 2005, pp. 82–89.
3. Das G., Gunopulos D., and Mannila H. Finding similar time series. In Advances in Knowledge Discovery and Data Mining, 1st Pacific-Asia Conf., 1997, pp. 88–100.
4. Gaffney S. and Smyth P. Trajectory clustering with mixtures of regression models. In Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 63–72.
5. Hadjieleftheriou M., Kollios G., Gunopulos D., and Tsotras V.J. On-line discovery of dense areas in spatio-temporal databases. In Proc. 8th Int. Symp. Advances in Spatial and Temporal Databases, 2003, pp. 306–324.
6. Han J. and Kamber M. Data Mining: Concepts and Techniques. Morgan Kaufmann, 2000.
7. Kalnis P., Mamoulis N., and Bakiras S. On discovering moving clusters in spatio-temporal data. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 364–381.
8. Lee J.-G., Han J., and Whang K.-Y. Trajectory clustering: a partition-and-group framework. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 593–604.
9. Mamoulis N., Cao H., Kollios G., Hadjieleftheriou M., Tao Y., and Cheung D.W. Mining, indexing, and querying historical spatiotemporal data. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 236–245.
10. Tao Y., Faloutsos C., Papadias D., and Liu B. Prediction and indexing of moving objects with unknown motion patterns. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 611–622.
11. Tsoukatos I. and Gunopulos D. Efficient mining of spatiotemporal patterns. In Proc. 7th Int. Symp. Advances in Spatial and Temporal Databases, 2001, pp. 425–442.
12. Vlachos M., Gunopulos D., and Kollios G. Discovering similar multidimensional trajectories. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 673–684.

13. Zaki M.J. Spade: an efficient algorithm for mining frequent sequences. Machine Learning, 42(1/2):31–60, 2001.

Spatio-Temporal Data Reduction

- ▶ [Compression of Mobile Location Data](#)

Spatio-Temporal Data Types

RALF HARTMUT GÜTING
University of Hagen, Hagen, Germany

Synonyms

- [Data Types for Moving Objects](#)

Definition

Abstract data types to represent time dependent geometries, in particular continuously changing geometries, or *moving objects*. The most important types are *moving point* and *moving region*.

Key Points

A *moving point* represents an entity for which only the time dependent position is of interest. A *moving region* describes an entity for which the time dependent location as well as the shape and extent are relevant. For example, moving points could represent people, vehicles such as cars, trucks, ships or planes, or animals; moving regions could be hurricanes, forest fires, spread of epidemic diseases etc. Moving point data may be captured by GPS devices or RFID tags; moving region data may result from processing sequences of satellite images, for example. Geometrically, moving points or moving regions exist in a 3D (2D + time) space, if the movement is modeled within the 2D plane; for moving points this can be easily extended to 4D (3D + time).

Beyond the most relevant types of moving point and moving region, to obtain a closed system there are related time dependent data types, such as real-valued functions or time dependent boolean values. To have a uniform terminology these types are also called *moving real* and *moving bool*, respectively. Static spatial data types such as *point*, *line* or *region*, and standard data

types are also needed. The data types include suitable operations such as:

trajectory: $m\text{point} \rightarrow m\text{line}$	Projection of a moving point into the plane
inside: $m\text{point} \times m\text{region} \rightarrow m\text{bool}$	When is a moving point inside a moving region
distance: $m\text{point} \times m\text{point} \rightarrow m\text{real}$	Distance between a moving and a static point

Cross-references

► [Moving Objects Databases and Tracking](#)

Recommended Reading

- Erwig M., Gütting R.H., Schneider M., and Vazirgiannis M. Spatio-temporal data types: an approach to modeling and querying moving objects in databases. *GeoInformatica*, 3:265–291, 1999.

Spatio-Temporal Data Warehouses

YUFEI TAO¹, DIMITRIS PAPADIAS²

¹Chinese University of Hong Kong, Hong Kong, China

²Hong Kong University of Science and Technology, Hong Kong, China

Synonyms

[Spatio-temporal online analytical processing](#); [Spatio-Temporal OLAP](#)

Definition

Consider N regions R_1, R_2, \dots, R_N and a time axis consisting of discrete timestamps $1, 2, \dots, T$, where T represents the total number of recorded timestamps (i.e., the length of history). The position and area of a region R_i may vary along with time, and its extent at timestamp t is denoted as $R_i(t)$. Each region carries a set of measures $R_i(t).ms$, also called the *aggregate data* of $R_i(t)$. The measures of regions change asynchronously with their extents. In other words, the measure of R_i ($1 \leq i \leq N$) may change at a timestamp t (i.e., $R_i(t).ms \neq R_i(t-1).ms$), while its extent remains the same (i.e., $R_i(t) = R_i(t-1)$), and vice versa.

A *spatio-temporal data warehouse* stores the above information, and efficiently answers the *spatio-temporal window aggregate* query, which specifies an area q_R

and a time interval q_T of continuous timestamps. The goal is to return the aggregated measure $\text{Agg}(q_R, q_T, f_{agg})$ of all regions that intersect q_R during q_T according to some distributive aggregation function f_{agg} or formally:

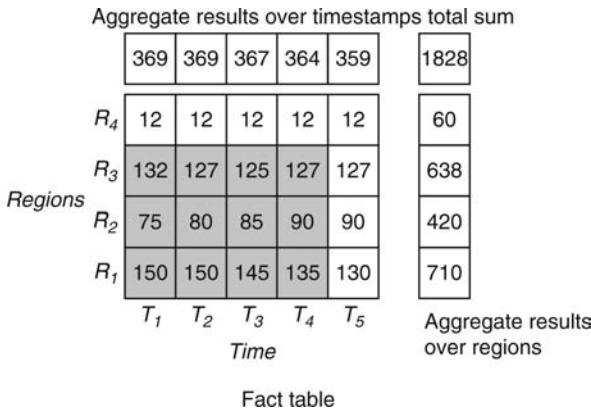
$$\begin{aligned}\text{Agg}(q_R, q_T, f_{agg}) &= f_{agg}\{R_i(t).ms \\ &| R_i(t) \text{ intersects } q_R \text{ and } t \in q_T\}.\end{aligned}$$

If q_T involves a single timestamp, the query is a *timestamp query*; otherwise, it is an *interval query*. An example of a timestamp window aggregate query is “find the total number of mobile users in the city center at 12 P.M.” The query will summarize the number of users (measures) in all regions intersecting $q_R = “city center”$ at $q_T = “12 P.M.”$

Historical Background

The motivation behind spatio-temporal data warehouses is that many spatio-temporal applications require summarized results, rather than information about individual objects. As an example, traffic supervision systems monitor the number of cars in an area of interest, instead of their ids. Similarly, mobile phone companies use the number of phone-calls per cell in order to identify trends and prevent potential network congestion. Although summarized results can be obtained using conventional operations on individual objects (i.e., accessing every single record qualifying the query), the ability to manipulate aggregate information *directly* is imperative in spatio-temporal databases due to several reasons. First, in some cases personal data should not be stored due to legal issues. For instance, keeping historical locations of mobile phone users may violate their privacy. Second, the individual data may be irrelevant or unavailable, as in the traffic supervision system mentioned above. Third, although individual data may be highly volatile and involve extreme space requirements, the aggregate information usually remains fairly constant for long periods, thus requiring considerably less space for storage.

A considerable amount of related research has been carried out on *data warehouses* and *OLAP* (on line analytical processing) in the context of relational databases. The most common conceptual model for data warehouses is the multi-dimensional data view. In this model, each *measure* depends on a set of *dimensions*, e.g., *region* and *time*, and thus is a value in the multi-dimensional space. A dimension is described by a



Spatio-Temporal Data Warehouses. Figure 1. A data cube example.

domain of values (e.g., days), which may be related via a hierarchy (e.g., day-month-year). Figure 1 illustrates a simple case, where each cell denotes the measure of a region at a certain timestamp. Observe that although regions are 2-dimensional, they are mapped as one dimension in the warehouse.

The *star schema* [6] is a common way to map a data warehouse onto a relational database. A main table (called *fact table*) F stores the multi-dimensional array of measures, while auxiliary tables D_1, D_2, \dots, D_n store the details of the dimensions. A tuple in F has the form $\langle D_i[], \text{key}, M[] \rangle$ where $D_i[].\text{key}$ is the set of foreign keys to the dimension tables and $M[]$ is the set of measures. OLAP operations ask for a set of tuples in F , or for aggregates on groupings of tuples. Assuming that there is no hierarchy in the dimensions of the previous example, the possible groupings in Fig. 1 include: (i) group-by Region and Time, which is identical to F , (ii)-(iii) group-by Region (Time), which corresponds to the projection of F on the *region-(time-)* axis, and (iv) the aggregation over all values of F which is the projection on the origin (Fig. 1 depicts these groupings for the aggregation function *sum*). The fact table together with all possible combinations of group-bys composes the *data cube* [2]. Although all groupings can be derived from F , in order to accelerate query processing some results may be pre-computed and stored as *materialized views*.

A detailed group-by query can be used to answer more abstract aggregates. In the example of Fig. 1, the total measure of all regions for all timestamps (i.e., 1828) can be computed either from the fact table, or by summing the projected results on the

time or *region* axis. Ideally, the whole data cube should be materialized to enable efficient query processing. Materializing all possible results may be prohibitive in practice as there are $O(2^n)$ group-by combinations for a data warehouse with n dimensional attributes. Therefore, several techniques have been proposed for the view selection problem in OLAP applications [1,4]. In addition to relational databases, data warehouse techniques have also been applied to spatial [3,10] and temporal [8] databases. All these methods, however, benefit only queries on a predefined hierarchy. An ad-hoc query not confined by the hierarchy, such as the one involving the gray cells in Fig. 1, would still need to access the fact table, even if the entire data cube were materialized.

Foundations

The next discussion describes several solutions to implementing a spatio-temporal data warehouse, assuming summation as the underlying aggregate function f_{agg} . Extensions to other aggregation functions (e.g., count, average) are straightforward.

- *Using a 3D aggregate R-tree*

The problem of a spatio-temporal window aggregate search can be regarded as a multi-dimensional aggregate retrieval in the 3D space (the spatial dimensions plus a time dimension) and solved using an aggregate R-tree (aR-tree). The aR-tree [5,9] is similar to a conventional *R-tree*, where each node also stores summarized information about the regions in each sub-tree. Whenever the extent or measure of a region changes, a new 3D box is inserted in a 3D version of the aR-tree, called the *a3DR-tree*. Using the example of Fig. 1, four entries are required for R_1 : one for timestamps 1 and 2 (when its measure remains 150) and three more entries for the other timestamps. A spatio-temporal window aggregate query can also be modeled as a 3D box, which can be processed on the a3DR-tree, following the strategy of solving a range aggregate query on an aR-tree [5,9].

The problem with this solution is that it creates a new box duplicating the region's extent, even though it does not change. Since the measure changes are much more frequent than extent updates, the *a3DR-tree* incurs high redundancy. The worst case occurs for static regions: although the extent of a region remains constant, it is still duplicated at the rate of its measure changes. Bundling the extent and aggregate information in all entries significantly lowers the node fanout

and compromises query efficiency, because more nodes must be accessed to retrieve the same amount of information. Note that redundancy incurs whenever the extent and measure changes are asynchronous, i.e., the above problem also exists when a new box is spawned because of an extent update, in which case the region's measure must be replicated.

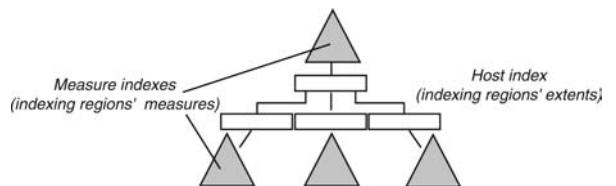
- *Using a data cube*

Following the traditional data warehouse approach, it is possible to create a data cube, where one axis corresponds to time, the other to regions, and keep the measure values in the cells of this two-dimensional table (see Fig. 1). Since the spatial dimension has no one-dimensional order, the table can be stored in the secondary memory ordered by time, and a B-tree index can be created to locate the pages containing information about each timestamp. The processing of a query employs the B-tree index to retrieve the pages (i.e., table columns) containing information about q_T ; then, these regions (qualifying the temporal condition) are scanned sequentially and the measures of those satisfying q_R are aggregated.

Even if an additional spatial index on the regions exists, the simultaneous employment of both indexes has limited effect. Assume that first a window query q_R is performed on the spatial index to provide a set of ids for regions that qualify the spatial condition. Measures of these regions must still be retrieved from the columns corresponding to q_T (which, again, are found through the B-tree index). However, the column storage does not preserve spatial proximity, and hence the spatially qualifying regions are expected to be scattered in different pages. Therefore, the spatial index has some effect only on very selective queries (on the spatial conditions). Furthermore, recall that pre-materialization is useless, since the query parameters q_R and q_T do not conform to pre-defined groupings.

- *The aggregate R-B-tree*

Since (i) the extent and measure updates are asynchronous and (ii) in practice, measures change much more frequently than extents (which may even be static), the two types of updates should be managed independently to avoid redundancy. This implies the deployment of two types of indexes: (i) a *host index*, which is an aggregate spatial or spatio-temporal structure managing region extents, and (ii) numerous *measure indexes* (one for each entry of the host index), which are

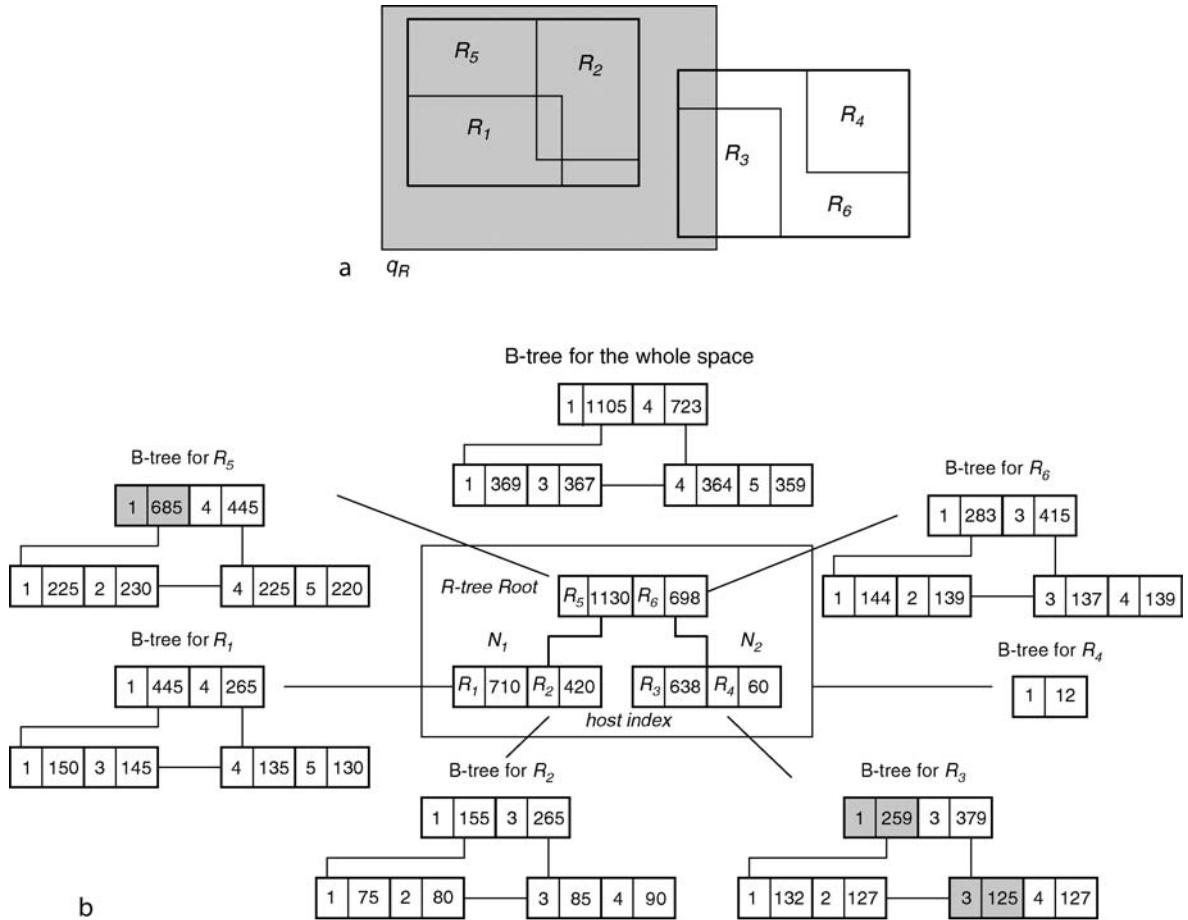


Spatio-Temporal Data Warehouses. Figure 2.

A multi-index architecture for indexing spatio-temporal data warehouses.

aggregate temporal structures storing the values of measures during the history. Figure 2 shows a general overview of the architecture [14]. Given a query, the host index is first searched, identifying the set of entries that qualify the spatial condition. The measure indexes of these entries are then accessed to retrieve the timestamps qualifying the temporal conditions. Since the number of records (corresponding to extent changes) in the host index is very small compared to the measure changes, the cost of query processing is expected to be low.

The following discussion explains an instantiation of the architecture for solving window aggregate queries when the underlying data regions are static. The instantiation leads to a structure, called the *aggregate R-B-tree (aRB-tree)*. It adopts an aR-tree as the host index, where an entry r has the form $\langle r.\text{MBR}, r.\text{aggr}, r.\text{pointer}, r.\text{btree} \rangle$; $r.\text{MBR}$ and $r.\text{pointer}$ have the same semantics as a normal R-tree, $r.\text{aggr}$ keeps the aggregated measure about r over the *entire history*, and $r.\text{btree}$ points to an aggregate *B-tree* which stores the detailed measure information of r at concrete timestamps. Figure 3b illustrates an example using the data regions of Fig. 3a and the measures of Fig. 1. The number 710 stored with R-tree entry R_1 , equals the sum of measures in R_1 for all 5 timestamps (e.g., the total number of phone calls initiated at R_1). The first leaf entry of the B-tree for R_1 (1, 150) indicates that the measure of R_1 at timestamp 1 is 150. Since the measure of R_1 at timestamp 2 is the same, there is no a special entry, but this knowledge is implied from the previous entry (1, 150). Similarly, the first root entry (1, 445) of the same B-tree indicates that the aggregated measure in R_1 during time interval [1,3] is 445. The topmost B-tree stores aggregated information about the whole space, and its role is to answer queries involving only temporal conditions (similar to that of the extra row in Fig. 1).



Spatio-Temporal Data Warehouses. Figure 3. A solution to static regions.

To illustrate the processing algorithms, consider the query “find the number of phone-calls initiated during interval $q_T = [1,3]$ in all cells intersecting the window q_R shown in Fig. 3a.” Starting from the root of the R-tree, the algorithm visits the B-tree of R_5 since the entry is totally contained in q_R . The root of this B-tree has entries (1,685), (4,445) meaning that the aggregated measures (of all data regions covered by R_5) during intervals [1,3], [4,5] are 685 and 445, respectively. Hence, the contribution of R_5 to the query result is 685. The second root entry R_6 of the R-tree partially overlaps q_R , so its child node is visited, where only entry R_3 intersects q_R , and thus its B-tree is retrieved. The first entry of the root (of the B-tree) suggests that the contribution of R_3 for the interval [1,2] is 259. In order to complete the result, the algorithm will have to descend the second entry and retrieve the measure of

R_3 at timestamp 3 (i.e., 125). The final result equals $685 + 259 + 125$, which corresponds to the sum of measures in the gray cells of Fig. 3b.

Key Applications

Traffic Control

Traffic control systems require summarized information about areas of interest instead of the concrete vehicle ids. Furthermore, in most cases, approximate aggregation [11] is sufficient for tasks such as traffic jam detection and shortest path computation.

Mobile Computing

Measures such as the number of phone-calls per cell can help identify trends, prevent potential network

congestion and achieve load balancing in mobile computing applications.

Sensor Systems

Spatio-temporal data warehouses can collect and store readings from geographically distributed sensors. As an example consider a pollution monitoring system, where the readings from several sensors are fed into a warehouse that arranges them in regions of similar or identical values. These regions should then be indexed for the efficient processing of queries such as “find the areas near the center with the highest pollution levels yesterday.”

Future Directions

Tao and Papadias [13] discuss spatial aggregation techniques, i.e., when both the regions and their measures are static. Tao et al. [12] present a sketch-based aggregation technique that avoids counting the same object twice (*distinct counting problem*) during the computation of aggregates. A survey of spatio-temporal aggregation techniques can be found in [7].

Experimental Results

[14] contains an extensive set of spatio-temporal aggregation techniques for static and dynamic regions, and experimental (as well as analytical) comparisons.

Data Sets

Common benchmark datasets can be found at:

www.rtreeportal.org

Cross-references

- ▶ [B+-Tree](#)
- ▶ [Data Warehouse](#)
- ▶ [On-Line Analytical Processing](#)
- ▶ [R-Tree \(and Family\)](#)
- ▶ [Star Schema](#)

Recommended Reading

1. Baralis E., Paraboschi S., and Teniente E. Materialized view selection in a multidimensional database. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 156–165.
2. Gray J., Bosworth A., Layman A., and Pirahesh H. Data cube: a relational aggregation operator generalizing group-by, cross-tabs and subtotals. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 152–159.
3. Han J., Stefanovic N., and Koperski K. Selective materialization: an efficient method for spatial data cube construction. In Proc.

Pacific-Asia Conf. on Knowledge Discovery and Data Mining, 1998, pp. 144–158.

4. Harinarayan V., Rajaraman A., and Ullman J. Implementing data cubes efficiently. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 205–216.
5. Jurgens M. and Lenz H. The Ra*-tree: an improved R-tree with materialized data for supporting range queries on OLAP-data. In Proc. Int. Workshop on Database and Expert Systems Applications, 1998, pp. 186–191.
6. Kimball R. The Data Warehouse Toolkit. John Wiley, New York, NY, 1996.
7. Lopez I., Snodgrass R., and Moon B. Spatiotemporal aggregate computation: a survey. IEEE Trans. Knowl. and Data Eng., 17(2):271–286, 2005.
8. Mendelzon A. and Vaisman A. Temporal queries in OLAP. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 242–253.
9. Papadias D., Kalnis P., Zhang J., and Tao Y. Efficient OLAP operations in spatial data warehouses. In Proc. 7th Int. Symp., Advances in Spatial and Temporal Databases, 2001, pp. 443–459.
10. Stefanovic N., Han J., and Koperski K. Object-based selective materialization for efficient implementation of spatial data cubes. IEEE Trans. Knowl. and Data Eng., 12(6):938–958, 2000.
11. Sun J., Papadias D., Tao Y., and Liu B. Querying about the past, the present and the future in spatio-temporal databases. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 202–213.
12. Tao Y., Kollios G., Considine J., Li F., and Papadias D. Spatio-temporal aggregation using sketches. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 214–225.
13. Tao Y. and Papadias D. Range aggregate processing in spatial databases. IEEE Trans. Knowl. and Data Eng., 16(12):1555–1570, 2004.
14. Tao Y. and Papadias D. Historical spatio-temporal aggregation. ACM Trans. Inf. Syst., 23(1):61–102, 2005.

Spatiotemporal Databases

- ▶ [Moving Objects Databases and Tracking](#)

Spatiotemporal Estimation

- ▶ [Spatiotemporal Interpolation Algorithms](#)

Spatiotemporal Graphs

- ▶ [Time Aggregated Graphs](#)

Spatiotemporal Interpolation Algorithms

PETER REVESZ

University of Nebraska-Lincoln, Lincoln, NE, USA

Synonyms

Moving objects interpolation; Spatio-Temporal approximation; Spatiotemporal estimation

Definition

Spatiotemporal interpolation is the problem of estimating the unknown values of some property at arbitrary spatial locations and times, using the known values at spatial locations and times where measurements were made. In spatiotemporal interpolation the estimated property varies with both space and time, with the assumption that the values are closer to each other with decreasing spatial and temporal distances.

Spatiotemporal interpolation is used in spatiotemporal databases, which record spatial locations and time instances together with other attributes that are dependent on space and time. For example, a spatiotemporal database may record the sales of houses in a town. The house sales database records the location, usually as the address of the house from which an (x, y) location can be easily found, by correlating the address with a map of the town, the calendar date when the sale occurred, the area and the sale price of the house.

Spatiotemporal interpolation is also used in moving objects databases to estimate the trajectory of moving objects.

Historical Background

Spatiotemporal interpolation is a generalization of spatial interpolation by the addition of a temporal dimension. While spatial interpolation is well-investigated [1], spatiotemporal interpolation in general, including interpolation of the trajectory of moving objects, is a relatively new area [5].

Spatiotemporal interpolation problems were either assumed to be just a sequence of spatial interpolation problems, called the *time slices approach*, or they were assumed to be as easily handled as adding one more spatial dimension, called the *extension approach*. Li and

Revesz [10] pointed out problems with the time slices and the extension approaches, and proposed the *spatiotemporal product (ST-product)* approach. Gao and Revesz [2] described several *adaptive spatiotemporal interpolation* approaches that improve the ST-product method.

Moving objects carrying positioning devices, thus recording their positions in arbitrary spatial locations along with their corresponding timestamps, provide data for moving object databases. In moving object databases a variety of spatiotemporal interpolation methods can be applied, e.g., linear interpolation [1], which is the most commonly adopted, and polynomial functions and splines [6,8,11,16].

Foundations

There are several problems with the simple spatiotemporal interpolation approaches, like the time slices approach and the extension approach. Among these problems the following need to be mentioned.

Poor Estimation Accuracy: The time slices approach is not accurate in general. For example, a spatiotemporal interpolation problem is to estimate the price of houses based on sales data from sold houses in a town. To solve this problem the time slices approach would first select just the houses that were sold in one short time slice, for example on a single day or in a single week. Then the time slices approach would do a spatial interpolation on the selected data. The accuracy of this method is often poor, because there is simply not too many houses sold on a single day or in a single week to cover the town dense enough for accurate interpolation. Many town subdivisions may not have a single house sold on just one day.

Lack of Continuous Time: The time slices approach cannot deal with continuous time.

Non-Invariance to Scaling: The extension approach handles a k -dimensional spatial and one dimensional temporal interpolation problem as if it were a $k+1$ -dimensional spatial interpolation problem. Li and Revesz [10] pointed out that the extension approach can also lead to problems because of the *non-invariance to scaling* of many spatial interpolation algorithms. Scaling invariance means that if the unit of measurement changes in one dimension, then the estimated value does not change. While many spatial interpolation algorithms are scaling invariant if the units

change in each spatial dimension, few spatial interpolation algorithms are scaling invariant if the units change only in one dimension. For example, the inverse distance weighting (IDW) [14] spatial interpolation method is non-invariant to scaling in only one dimension. Such methods are difficult to use because it is difficult to decide what is the right unit of time given certain units of spatial distance. For example, IDW gives a different house price estimate if the unit of time is days than if the unit of time is weeks, even if all houses are always sold on Mondays. That is a strange phenomenon that makes IDW inherently awkward to use.

The spatiotemporal product (or ST-product) approach [10] improves the times slices method. Using the house price estimation problem, the ST-product approach can be explained as follows. For each house that is sold several times according to the sales spatiotemporal database, the ST-product approach first estimates its price based on a simple linear temporal interpolation between two consecutive sales. For example, if a house was sold for \$200,000 on July 29th, 2000, and was sold for \$250,000 on July 1st, 2002, then one can estimate that the price of the house rose \$500 for each of the 100 weeks between the two sales.

After these temporal interpolations, a spatial interpolation is done for each week like in the time slices method. However, since a large percentage of the values for each week are filled in, the ST-product method uses a considerably greater density of houses in the spatial interpolation part than the time-slices method does. Hence, in general, the ST-product method is more accurate than the time-slices method. Further, the ST-product method can be applied without an exact calculation of the temporal interpolation values by using a temporal parameter t . Then the ST-product method yields an interpolation function of x , y , and t for any (x, y) location and time instance t , even in continuous time.

The spatiotemporal accuracy is increased further by *adoptive spatiotemporal interpolation* [2]. To explain the adoptive method, suppose $R(x, y, t, w)$ is a spatiotemporal relation where (x, y) is the location, t is the time instance, and w is the measured value. The key idea behind the adaptive method is that to interpolate the value w at any location (a, b) at time c , where a, b and c are constants, there are two main choices:

1. *Spatial Projection + 1D temporal interpolation*: Select from R the records with $x = a$ and $y = b$. Then use any 1D temporal interpolation method on the selected records.
2. *Temporal Projection + 2D spatial interpolation*: Select from R the records with $t = c$. Then use any 2D spatial interpolation method on the selected records.

When the spatial and temporal projections of R both contain enough number of records to do an interpolation at location (a, b) and time c , then one could choose either (1) or (2) as an option. These two options may give different interpolation values. Which one of the two estimates is more reliable? The *choice-based adaptive interpolation method* decides that question based on the *relationship strength measures* for space and time. Intuitively, the larger the relationship strength measure for space (or time), the more reliable is the estimation using a spatial (or temporal) interpolation. These measures are denoted by the following symbols:

$\mathcal{S}(a, b, c)$ – the relationship strength measure for space at location (a, b) and time c .

$\mathcal{T}(a, b, c)$ – the relationship strength measure for time at location (a, b) and time c .

Both measures are localized, hence the spatial relationship strength can be larger than the temporal relationship strength at some locations and times, while the reverse may be true at a different location and time within the same problem. The choice-based adaptive method works as follows whenever there is a choice:

$$\begin{cases} \text{Spatial interpolation} & \text{if } \mathcal{S}(a, b, c) > \mathcal{T}(a, b, c) \\ \text{Temporal interpolation} & \text{if } \mathcal{S}(a, b, c) \leq \mathcal{T}(a, b, c) \end{cases} \quad (1)$$

Both the ST-product and the choice-based adaptive methods first interpolate the missing values for the (a, b) locations for which several measurements at different times are known. While the ST-product method always preferred to do a temporal interpolation, the adaptive method at some spatiotemporal locations will prefer to do a spatial interpolation. This is the only reason why the two interpolation results may disagree. Finally, for both methods the

non-measurement locations, that is, the locations that do not appear in the original data set, need to be estimated using some 2D spatial interpolation.

Gao and Revesz [2] suggested several spatial and temporal relationship strength measures. A simple measure is to use the inverse of the variance of the measured values in the spatial neighborhood of (a, b) , which is defined as the nearest k other (a, b) locations for some integer k , as the spatial relationship strength. Similarly, one can use the inverse of the variance of the two measures at the earliest time following c , and latest time preceding c , as the temporal relationship strength. As an alternative to the choice-based adaptive interpolation approach, [2] also proposed a linear combination adaptive interpolation approach, which gives a weighted linear sum of the two estimates. The weights depend on the relative magnitudes of the spatial and temporal relationship strength measures.

Trajectory of Moving Objects: The estimation of the trajectory of moving objects can also be viewed as a spatiotemporal interpolation problem. Applying the above described ideas, in each sensor location the moving object can be either sensed with value one or not sensed and assigned a value zero. However, there are more advanced interpolation techniques, which associate each moving object's approximated position with an uncertainty factor [9,12,15].

Key Applications

Spatiotemporal interpolation has a growing number of applications, in areas such as the following:

1. *Real Estate Analysis:* In an experiment with a database of house sales in the town of Lincoln, Nebraska, USA, the ST-product method had less than a 9% mean absolute error in estimating house prices [10]. The surprising feature of the estimation was that nothing special was known about the houses except their locations, sizes in square feet, and prices. The estimation could be easily improved with a visit to the houses to check their conditions. The house price estimation can tell whether some given houses in given years were assessed too high or too low taxes.
2. *Weather Analysis:* Weather is a spatiotemporal phenomenon that shows several cyclical patterns. The ST-product spatiotemporal interpolation method was used for ground water level and drought

analysis. Carbon dioxide concentrations over time is another application area.

3. *Epidemiology:* Spatiotemporal interpolation was also used to predict the spread of epidemics, for example, the spread of the West Nile Virus in the continental USA [13].
4. *Forest Fires:* The spread of forest fires can be predicted similar to the spread of epidemics.
5. *Traffic Accident Report:* In a traffic accident report spatiotemporal interpolation may be used to estimate the trajectories of the vehicles that were involved in the accident. An accurate estimation may be important to decide what caused the accident and to get insurance payments.

Future Directions

There are still several problems about the relationship of spatiotemporal interpolation and prediction of spatiotemporal phenomena. Spatiotemporal interpolation seems to work best if one is interested in a location that is in the middle of the space, and in a time that is in the middle of the time interval recorded. Predictions are at future times, hence they use specialized algorithms. For example, voting prediction is a research area in itself and has many specialized techniques different from the spatiotemporal interpolation approaches. For example, when one is trying to predict the outcome of an election in a voting district A , one cannot use the time-slices, the ST-product, and the adoptive interpolation approaches because there are no spatial neighbors in general. There are some exceptions to this rule. For example, if the neighboring voting districts have all already reported their election results while the votes in district A are still being (re)counted, then one can predict the outcome in voting district A using the results of the neighboring districts [3].

While voting problems are almost always prediction problems, on occasion, analysis of past election results may be also interesting, for example, to identify and help settle possible election fraud cases in the past. Predicting other spatiotemporal behaviors beside voting, for example, the number of customers in a town who would buy a certain product is also an important area of interest.

Spatiotemporal databases can be conveniently represented using *constraint databases* [7]. Constraint databases allow convenient handling of different types of interpolation data [4].

Cross-references

- ▶ Constraint Databases
- ▶ Databases
- ▶ Moving Object
- ▶ Moving Objects Databases and Tracking

Recommended Reading

1. Davis P.J. Interpolation and Approximation. Dover, NY, USA, 1975.
2. Gao J. and Revesz P. Adaptive spatiotemporal interpolation methods. In Proc. First Int. Conf. on Geometric Modeling, Visualization, and Graphics, 2005, pp. 1622–1625.
3. Gao J. and Revesz P. Voting prediction using new spatiotemporal interpolation methods. In Proc. 7th Int. Conf. on Digital Government Research, 2006, pp. 293–300.
4. Grumbach S., Rigaux P., and Segoufin L. Manipulating interpolated data is easier than you thought. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 156–165.
5. Güting R. and Schneider M. Moving Objects Databases. Morgan Kaufmann, Los Altos, CA, 2005.
6. Hadjieleftheriou M., Kollios G., Tsotras V.J., and Gunopulos D. Efficient indexing of spatiotemporal objects. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 251–268.
7. Kanellakis P.C., Kuper G.M., and Revesz P. Constraint query languages. *J. Comput. Syst. Sci.*, 51(1):26–52, 1995.
8. Koubarakis M., Sellis T.K., Frank A.U., Grumbach S., Güting R. H., Jensen C.S., Lorentzos N.A., Manolopoulos Y., Nardelli E., Pernici B., Schek H.-J., Scholl M., Theodoulidis B., and Tryfona N. (eds.). Spatio-Temporal Databases: The CHORO-CHRONOS Approach, LNCS 2520. Springer, 2003.
9. Kuijpers B. and Othman W. Trajectory databases: Data models, uncertainty and complete query languages. In Proc. 11th Int. Conf. on Database Theory, 2007, pp. 224–238.
10. Li L. and Revesz P. Interpolation methods for spatiotemporal geographic data. *J. Comput. Environ. Urban Syst.*, 28(3):201–227, 2004.
11. Ni J. and Ravishankar C.V. Indexing spatio-temporal trajectories with efficient polynomial approximations. *IEEE Trans. Knowl. Data Eng.*, 19(5):663–678, 2007.
12. Pfoser D. and Jensen C.S. Capturing the uncertainty of moving-object representations. In Proc. Int. Symp. on Large Spatial Databases, 1999, pp. 111–132.
13. Revesz P. and Wu S. Spatiotemporal reasoning about epidemiological data. *Artif. Intell. Med.*, 38(2):157–170, 2006.
14. Shepard D.A. A two-dimensional interpolation function for irregularly spaced data. In Proc. 23rd ACM National Conf., 1968, pp. 517–524.
15. Trajcevski G., Wolfson O., Zhang F., and Chamberlain S. The geometry of uncertainty in moving objects databases. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 233–250.
16. Yu B., Kim S.H., Bailey T., and Gamboa R. Curve-based representation of moving object trajectories. In Proc. 8th Int. Database Engineering and Applications Symp., 2004, pp. 419–425.

Spatio-Temporal OLAP

- ▶ Spatio-Temporal Data Warehouses

Spatio-Temporal Online Analytical Processing

- ▶ Spatio-Temporal Data Warehouses

Spatio-Temporal Representation

- ▶ Spatio-Temporal Trajectories

Spatio-Temporal Selectivity Estimation

GEORGE KOLLIOS
Boston University, Boston, MA, USA

Synonyms

Selectivity for predictive spatio-temporal queries

Definition

In spatio-temporal databases, the locations of moving objects are usually modeled as linear functions of time. Thus, the location of an object at time t is represented as $o(t) = o_s + o_v t$, where o_s is the initial location of the object at time $t = 0$ and o_v is its velocity. Given that the object moves in a d -dimensional space, $o(t)$, o_s , and o_v are d -dimensional vectors. In this setting, the selectivity estimation of spatio-temporal queries is defined as follows:

Given a database that stores the locations of moving objects and a spatio-temporal query, estimate the number of objects that satisfy the query.

There are two important types of queries in this environment: spatio-temporal window (or range) queries and spatio-temporal distance join queries. A spatio-temporal window query (STWQ) specifies a (static or moving) region q_S , a future time interval q_T , and a

dataset of moving objects D , and retrieves all data objects that will intersect (or will be covered by) q_S during q_T . On the other hand, a spatio-temporal distance join query (STDJQ) assumes two datasets D_A and D_B of moving objects, a time period q_T and a distance q_d and asks for the pairs of objects (o_a, o_b) that $o_a \in D_A, o_b \in D_B$ and are closer than q_d during the time interval q_T .

Historical Background

The problem of spatio-temporal selectivity estimation is related to spatio-temporal indexing and spatial selectivity estimation. In the spatio-temporal indexing problem the system must report all the objects that satisfy the query; here the system must just give an estimate on the number of such objects. In spatial selectivity estimation the main difference is that the objects are static and therefore easier to model and represent succinctly.

In particular, selectivity estimation for spatial queries is based on spatial histograms. A histogram partitions the data space into a set of buckets, and the object distribution in each bucket is assumed to be (almost) uniform. A bucket B stores the number $B.num$ of objects whose centroids fall in B , and the average extent $B.len$ of such objects. Consider a window query Q . Then, using an analysis based on uniformly distributed data [12,7], the expected number of qualifying objects in B is approximated by $B.num \frac{I.area}{B.area}$, where $I.area$ is the area of the intersection of the query Q with bucket B and $B.area$ the area of B . The total number of objects intersecting Q is estimated by summing the results of all buckets. Evidently, satisfactory estimation accuracy depends on the degree of uniformity of objects' distributions in the buckets. An example of a histogram construction that generates nearly uniform buckets for spatial datasets appeared in [1].

Another approach uses spatial sketches to estimate the selectivity of spatial queries and in addition provides probabilistic guarantees on the quality of the estimation [4]. In particular, given a dataset D of n spatial rectangles (points are assumed to be degenerated rectangles) it is possible to create a synopsis for each dataset, which has size poly-logarithmic to n and proportional to $\frac{1}{\epsilon}$, and provides answers that with high probability are $\pm \epsilon$ away from the exact answer. The synopsis is based on AGMS sketches [2] extended to handle multidimensional intervals (rectangles) in poly-logarithmic space. Note that the histogram based techniques provide no guarantees on

the estimation quality (e.g., in the worst case the error can be arbitrary large), but work well in most practical cases.

Foundations

To estimate the selectivity of STWQs efficiently two basic approaches have been proposed. One is based on random sampling and the other on spatio-temporal histograms. Moreover, the histogram based approach can be extended for STDJQs as well.

Selectivity Estimation for STWQ

The simplest approach is to use random sampling [6]. Given a dataset of n moving objects the method keeps a uniform random sample S of the set of moving objects that is used to estimate the result. Note that each time an object issues an update, the function that represents its location in the database must change. Therefore, a tuple must be deleted (the one that corresponds to the old function) and a new one must be inserted. To maintain a uniform random sample in such a dynamic environment a specialized solution must be used [5,9]. For a query Q , let r be the size of the result of Q over the random sample S . Then, the method estimates the result of Q as $r \frac{n}{|S|}$, where $|S|$ is the size of S . If the set of queries is known beforehand, then a technique that is based on stratified sampling can be applied which reduces the size of the sample and increases the accuracy of the estimator. This improved method is called Venn sampling and appeared in [10].

The other approach is based on spatio-temporal histograms. For d -dimensional moving points, the histogram is constructed in a $2d$ -dimensional space, where d dimensions represent the coordinates of the moving objects at a reference time instant (e.g., $t = 0$) and the other d dimensions their velocity. Each bucket stores the number of objects and the minimum and maximum locations and velocities of these objects in each dimension. Furthermore, it is assumed that the objects are distributed uniformly inside the bucket. Based on that, a query estimation procedure is used to find the percentage of the objects in each bucket that are expected to intersect the query. Then, the (approximate) result of Q is obtained by summing the contributions of all buckets (see [3,6,11] for more details). The spatio-temporal histogram can also be dynamically maintained in the presence of object updates.

Selectivity Estimation for STDJQ

For estimating the size of a STDJQ between two sets of moving objects D_A and D_B , two spatio-temporal histograms are maintained, one for each set. Then, there are two approaches to estimate the size of the join. In one approach, the distance between objects is defined using the L_{max} norm, i.e., given two d -dimensional points a and b , their distance is: $dist(a, b) = \max_{i=1,2,\dots,d} |a.x_i - b.x_i|$. Because of the independence between the dimensions under this distance, the selectivity of the join query can be estimated using the selectivity of each dimension independently of the others. Thus, the selectivity of a STDJQ Q is expressed as $Sel(Q) = \prod_{i=1}^d Sel_i(Q)$. Furthermore, the $Sel_i(Q)$ can be estimated using the projections of the buckets to dimension i . More details can be found in [8].

In the other approach, the distance function is the L_2 metric (i.e., Euclidean distance), and a more complicated formula is used to estimate the selectivity of the query. The analysis is based on the following idea: consider a moving object p_1 with specific location and velocity and another object p_2 with fixed velocity. Also, assume that the location of p_2 is distributed uniformly in space. Based on these assumptions, it is possible to compute the probability that p_2 will satisfy the query. Moreover, using this probability, the selectivity of the SPDJQ can be estimated. Note that, this method can be used for both moving points and moving rectangles and can be extended to other L_p norms. The details of this technique appears in [11].

Key Applications

Selectivity estimation is used extensively in database query optimization. Therefore, the techniques developed for spatio-temporal selectivity estimation can be useful in systems that need to generate execution plans for spatio-temporal queries. Also, many application like traffic monitoring, sensor networks or mobile communications, require aggregate information about the locations of moving objects. Furthermore, getting the exact answer to these queries can be very expensive. In that case, fast approximate answers on spatio-temporal aggregation queries using the techniques discussed above is the best alternative.

Future Directions

Current techniques for spatio-temporal selectivity estimation use the assumption that objects move

linearly over time. However, in many real life application this may not be a good approximation. It is an open problem how to extend the existing techniques to handle moving objects with non-linear motion functions.

Cross-references

- [Indexing of the Current and Near-Future Positions of Moving Objects](#)
- [Spatio-Temporal Data Warehouses](#)

Recommended Reading

1. Acharya S., Poosala V., and Ramaswamy S. Selectivity estimation in spatial databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 13–24.
2. Alon N., Gibbons P.B., Matias Y., and Szegedy M. Tracking join and self-join sizes in limited storage. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 10–20.
3. Choi Y.-J. and Chung C.-W. Selectivity estimation for spatio-temporal queries to moving objects. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 440–451.
4. Das A., Gehrke J., and Riedewald M. Approximation techniques for spatial data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 695–706.
5. Frahling G., Indyk P., and Sohler C. Sampling in dynamic data streams and applications. In Proc. Symp. on Computational Geometry, 2005, pp. 142–149.
6. Hadjieleftheriou M., Kollios G., and Tsotras V.J. Performance evaluation of spatio-temporal selectivity estimation techniques. In Proc. 15th Int. Conf. on Scientific and Statistical Database Management, 2003, pp. 202–211.
7. Pagel B.-U., Six H.-W., Toben H., and Widmayer P. Towards an analysis of range query performance in spatial data structures. In Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1993, pp. 214–221.
8. Sun J., Tao Y., Papadias D., and Kollios G. Spatio-temporal join selectivity. Inf. Syst., 31(8):793–813, 2006.
9. Tao Y., Lian X., Papadias D., and Hadjieleftheriou M. Random sampling for continuous streams with arbitrary updates. IEEE Trans. Knowl. Data Eng., 19(1):96–110, 2007.
10. Tao Y., Papadias D., Zhai J., and Li Q. Venn sampling: a novel prediction technique for moving objects. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 680–691.
11. Tao Y., Sun J., and Papadias D. Analysis of predictive spatio-temporal queries. ACM Trans. Database Syst., 28(4):295–336, 2003.
12. Theodoridis Y. and Sellis T. A model for the prediction of R-tree performance. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1996, pp. 161–171.

Spatio-Temporal Stream Processing

- [Continuous Monitoring of Spatial Queries](#)

Spatio-Temporal Trajectories

ELIAS FRENTZOS¹, YANNIS THEODORIDIS¹,

APOSTOLOS N. PAPADOPOULOS²

¹University of Piraeus, Piraeus, Greece

²Aristotle University, Thessaloniki, Greece

Synonyms

Moving object trajectories; Spatio-temporal representation

Definition

A spatio-temporal trajectory can be straightforwardly defined as a function from the temporal $I \subseteq \mathbb{R}$ domain to the geographical space \mathbb{R}^2 , i.e., the 2-dimensional plane. From an application point of view, a trajectory is the recording of an object's motion, i.e., the recording of the positions of an object at specific timestamps.

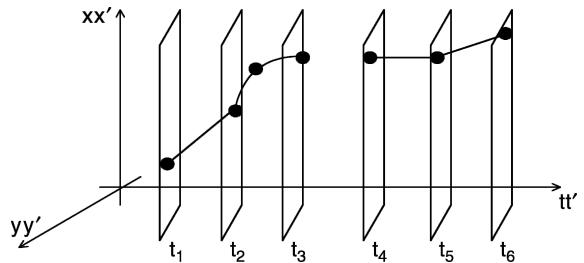
Generally speaking, spatio-temporal trajectories can be classified into two major categories, according to the nature of the underlying spatial object: (i) objects without area represented as moving points, and (ii) objects with area, represented as moving regions; in this case the region extent may also change with time. Among the above two categories, the former has attracted the main part of the research interest, since the majority of real-world applications involving spatio-temporal trajectories consider objects represented as points, e.g., fleet management systems monitoring cars in road networks.

Focusing on trajectories of moving points, while the actual trajectory consists of a curve, real-world requirements imply that the trajectory has to be built upon a set of sample points, i.e., the time-stamped positions of the object. Thus, trajectories of moving points are often defined as sequences of (x, y, t) triples:

$$T = \{(x_1, y_1, t_1), (x_2, y_2, t_2), \dots, (x_n, y_n, t_n)\},$$

where $x_i, y_i, t_i \in \mathbb{R}$, and $t_1 < t_2 < \dots < t_n$ and the actual trajectory curve is approximated by applying *spatio-temporal interpolation* methods on the set of sample points; among the proposed in the literature *spatio-temporal interpolation* techniques, the notion of linear interpolation has been widely adopted, given that it is fast, natural, and easy to implement (Fig. 1).

There are several techniques developed merely for the management of spatio-temporal trajectories in databases: *spatio-temporal models and languages*,



Spatio-Temporal Trajectories. Figure 1. The spatio-temporal trajectory of a moving point: dots represent sampled positions and lines in between represent alternative interpolation techniques (linear vs. arc interpolation). Unknown type of motion can be also found in a trajectory (see $[t_3, t_4]$ time interval).

indexing of historical spatio-temporal data, advanced query processing, and trajectory summarization techniques.

Historical Background

From a modeling perspective, the concept of spatio-temporal trajectories was introduced in some early works [6,7,10], which addressed the need for capturing the complete history of objects' movement. Clearly, as location data may change over time, the database must contain the whole history of this development. Thus, the DBMS should be allowed to go back in time at any particular timestamp, and to retrieve the state of the database at that time. More specifically, [11] models moving points (*mpoints*) and moving regions (*mregions*) as 3-dimensional (2D space + time) or higher-dimensional entities whose structure and behavior is captured by modeling them as abstract data types. Such types and their operations for spatial values changing over time can be integrated as base (attribute) data types into an extensible DBMS. Guting et al. [11] introduced a type constructor τ which transforms any given atomic data type a into a type $\tau(a)$ with semantics $\tau(a) = \text{time} \rightarrow a$. In this way, the two aforementioned basic types, namely *mpoint* and *mregion*, may be also represented as $\tau(\text{point})$ and $\tau(\text{region})$, respectively. Guting et al. [11] also provided an algebra with data types (such as moving point, moving region, moving real, etc.) together with a comprehensive set of operations, supporting a variety of queries of spatio-temporal trajectory data.

On another line of research, [14] first dealt with the special requirements that spatio-temporal trajectories

Spatio-Temporal Trajectories. **Table 1.** Classification of spatio-temporal queries

Query type		Operation
Coordinate-based		<i>Overlap, inside, etc.</i>
Trajectory-based	Topological	<i>Enter, leave, cross, bypass, etc.</i>
	Navigational	<i>Travelled distance, covered area, speed, heading, etc.</i>

pose to the database engine, in terms of efficient index structures and specific query processing techniques. In particular, [14] addressed the most commonly used queries over spatio-temporal trajectories and classified them according to [Table 1](#). Based on the observation that many query types are trajectory-based (i.e., they require the knowledge of a significant part of the moving objects' trajectory), [14] proposed the Trajectory Bundle tree (TB-tree), based on the well-known *R-tree*, considered as a seminal work in the context of *Indexing Historical Spatio-temporal Data*.

Foundations

Spatio-temporal trajectories may be queried with a variety of operators, which are mainly extensions of existing spatial operators. Among them, the simple spatio-temporal range query, involving both spatial and temporal components over *R-tree* structures indexing spatio-temporal trajectories, is a straightforward generalization of the standard *R-tree FindLeaf* algorithm in the 3-dimensional space. *Nearest neighbor queries* have been also considered in the context of spatio-temporal trajectories; the algorithms over *R-trees* and variations (e.g., TB-tree) proposed in [8], are based on both depth-first and best-first *R-tree* traversals, similar to the algorithms used for nearest neighbor querying over spatial data. The proposed algorithms vary with respect to the type of the query object (stationary or moving point) as well as the type of the query result (historical continuous or not), thus resulting in four types of nearest neighbor queries.

Trajectory join has been also investigated in several papers motivated as an extension of the respective spatial operator. Bakalov et al. in [3] consider the problem of evaluating all pairs of trajectories between two datasets, during a given time interval, which, given a distance function, all distances between timely

corresponding trajectory positions are within a given threshold. Then an approximation technique is used to reduce the trajectories into symbolic representations (strings) so as to lower the dimensionality of the original (3-dimensional) problem to one. Using the constructed strings, a special lower-bounding metric supports a pruning heuristic which reduces the number of candidate pairs to be examined. The overall schema is subsequently indexed by a structure based on the *B-tree*, requiring also minimal storage space. Another variation on the subject of joining trajectories is the closest-point-of-approach recently introduced in [2]. Closest-point-of-approach requires finding all pairs of line segments between two trajectories such that their distance is less than a predefined threshold. The work presented in [2] proposes three approaches; the first utilizes packed *R-trees* treating trajectory segments as simple line segments in the $d + 1$ dimensional space, and then employs the well known *R-tree spatial join* algorithm which requires carefully controlled synchronized traversal of the two *R-trees*. The second is based on a plane sweep algorithm along the temporal dimension, while the third is an adaptive algorithm which naturally alters the way in which it computes the join in response to the characteristics of the underlying data.

Much more challenging are the so called *similarity-based* queries over spatio-temporal trajectory data. Similarity search has been extensively studied within the time series domain; consequently, techniques addressed there, are usually extended in the spatio-temporal domain, in which spatio-temporal trajectories are considered as time series. Traditionally, similarity search has been based on the Euclidean Distance between time series, nevertheless having several disadvantages which the following proposals are trying to confront. In particular, in order to compare sequences with different lengths, [12] use the Dynamic Time Warping (DTW) technique that allows sequences to be stretched along the time axis so as to minimize the distance between sequences. Although DTW incurs a heavy computation cost, it is robust against noise. Moreover, in order to reduce the effect of its quadratic complexity on large time series, a lower bounding function along with a dedicated index structure has been proposed for pruning in [12]. Longest Common SubSequence (LCSS) measure [16] matches two sequences by allowing them to stretch, without rearranging the sequence of the elements, also allowing some elements to be

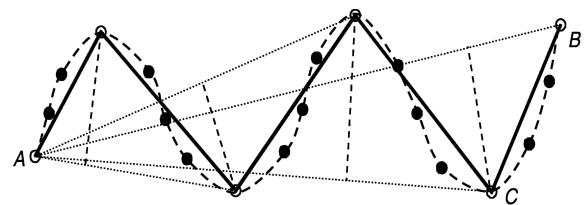
unmatched (which is the main advantage of the LCSS measure compared to Euclidean Distance and DTW). Therefore, LCSS can efficiently handle outliers and different scaling factors. In [5], a distance function, called Edit Distance on Real Sequences (EDR), was introduced. EDR distance function is based on the edit distance, which is the number of insert, delete, or replace operations that are needed to convert a trajectory T into Q . In the respective experimental study presented in [5], EDR was shown to be more robust than DTW and LCSS over trajectories with noise. In order to speed up the similarity search between trajectories, both [16] and [5], rely on dedicated index structures, thus achieving pruning of over 90% of the total number of indexed trajectories. However, such approaches fail to utilize the most commonly available access methods for *Indexing Historical Spatio-temporal Data* such as *R-trees*, leading to additional overhead.

In order to overcome this disadvantage, [9] employ the average Euclidean distance between two trajectories as a measure of their dissimilarity, and then, using *R-trees* indexing spatio-temporal trajectories provide a series of metrics and heuristics which efficiently prune the search space. These metrics are based (i) on the observation that an upper value for the speed of the spatio-temporal trajectories can provide lower and upper bounds of the average Euclidean distance between two trajectories, and (ii), on the fact that a best-first *R-tree* traversal on the $\text{mindist}(N, q)$ between a query trajectory q and a node N , provides a tighter lower bound for the average Euclidean distance. Finally, [9] provide an efficient algorithm for k -most similar trajectory search over *R-trees* indexing spatio-temporal trajectories.

In the indexing domain, a challenging line of research deals with network-constrained trajectories; this is due to the fact that the majority of the applications involving spatio-temporal trajectories deal with objects moving along *road networks* (i.e., cars, buses, trains). Following this observation, several specific access methods for objects moving in networks have been proposed; among them the most efficient is the Moving Objects in Networks tree (MON-tree) [1]. However, in order to exploit such network constrained indexes, the need for mapping the trajectories into the underlying network introduces the so called *map matching* problem. Specifically, the observation that raw trajectory positions are affected by the

measurement error introduced by e.g., GPS, and, the sampling error being up to the frequency with which position samples are taken, reveals the problem of correctly matching such tracking data in an underlying map containing e.g., a *road network*. Currently, the state-of-the-art approach addressing the map matching problem is the one presented in [4], which proposes mapping the entire trajectory to candidate paths in the *road network* using the Fréchet distance, which can be illustrated as follows: suppose a human and her dog constrained to walk on two different curves, while they are both allowed to control their speed independently. Then, the Fréchet distance between the two curves is the minimal length of a leash that is necessary. The proposed global map-matching algorithms in [4] find a curve in the *road network* that is as close as possible to the given trajectory in terms of the Fréchet distance between them.

Last but not least, it is the need for compression techniques that arises due to the fact that all the ubiquitous positioning devices will eventually start to generate an unprecedented stream of time-stamped positions, leading to storage and computation challenges [13]. In this direction, [13] exploit existing algorithms used in the line generalization field, and present one *top-down* and one *opening window* algorithm, which can be directly applied to spatio-temporal trajectories. The *top-down* algorithm, named TD-TR, is based on the well known Douglas-Peucker algorithm (Fig. 2) originally used in the context of cartography. This algorithm calculates the perpendicular distance of each internal point from the line connecting the first and the last point of the polyline (line AB in Fig. 2) and finds the point with the greatest perpendicular distance (point C). Then, it creates lines AC and CB

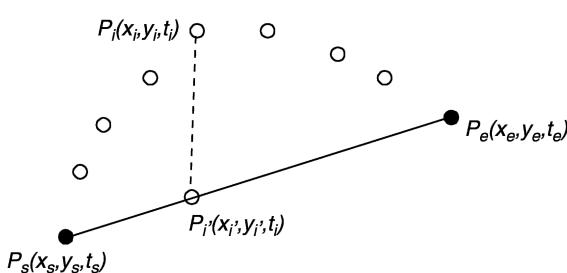


Spatio-Temporal Trajectories. Figure 2. Top-down Douglas-Peucker algorithm used for trajectory compression. Original data points are represented by closed circles [MB04] [13].

and, recursively, checks these new lines against the remaining points with the same method. When the distance of all remaining points from the currently examined line is less than a given threshold (e.g., all the points following C against line BC in Fig. 2) the algorithm stops and returns this line segment as part of the new – compressed – polyline. Being aware of the fact that trajectories are polylines evolving in time, the algorithm presented in [13] replaces the perpendicular distance used in the DP algorithm with the so-called *Synchronous Euclidean Distance* (SED), which is the distance between the currently examined point (P_i in Fig. 3) and the point of the line (P_s, P_e) where the moving object would lie, assuming it was moving on this line, at time instance t_i determined by the point under examination (P'_i in Fig. 3). The experimental study presented in [13] shows that such compression techniques introduce a small and manageable error, reducing at the same time the size of the dataset under 40% of its original size.

Key Applications

- *Location-based Services (LBS)* – Spatio-temporal trajectories are used in location-based services for determining the exact position of users, based on the map-matching solutions provided over trajectories.
- *Spatio-temporal Decision Support Systems (STDSS)* – A number of decision support tasks can exploit the presence of spatio-temporal trajectories. Traffic estimation and prediction systems. Analysis of



Spatio-Temporal Trajectories. Figure 3. The Synchronous Euclidean Distance (SED): The distance is calculated between the point under examination (P_i) and the point P'_i which is determined as the point on the line (P_s, P_e) the time instance t_i [MB04] [13].

traffic congestion conditions. Fleet management systems. Urban and regional planers analyzing the life courses of city residents. Scientists studying animal immigration habits.

Future Directions

There are several research directions arising regarding spatio-temporal trajectory data management. For example, the problem of estimating the selectivity of a range query over historical trajectory data still remains open. More specifically, such a technique would have to deal with the *distinct counting* problem [15], which is also present in the context of *Spatio-temporal Data Warehouses*. This problem stands when an object samples its position in several timestamps inside a given query window resulting to be counted multiple times in the query result. Nevertheless, a selectivity estimation technique based on a space partitioning method, such as a histogram, would have to return the number of distinct trajectories contained inside the query region, summing the containment of several buckets; then trajectories appearing in several buckets would have to be counted only once.

Another interesting research direction appears when considering network-constraint trajectory compression; in particular, existing compression techniques do not consider that trajectories may be network-constraint, resulting in trajectories which after the compression may be invalid regarding the underline network. As such, future work should investigate on techniques which may produce compressed trajectories being still valid under the network constraints.

Experimental Results

In general, for every presented method, there is an accompanying experimental evaluation in the corresponding reference.

Data Sets

A collection of real spatio-temporal datasets, as well as links to generators for spatio-temporal trajectory data can be found at *R-tree portal* (URL: <http://www.rtreeportal.org/>).

Url to Code

R-tree portal (URL: <http://www.rtreeportal.org/>) contains the code for most common spatio-temporal

indexes, as well as data generators and several useful links on spatio-temporal databases.

Cross-references

- [Indexing Historical Spatio-Temporal Data](#)
- [Nearest Neighbor Query](#)
- [Road Networks](#)
- [Rtree](#)
- [Spatial](#)
- [Spatial and Spatio-Temporal Data Models and Languages](#)
- [Spatial Join](#)
- [Spatio-Temporal Data Warehouses](#)
- [Spatiotemporal Interpolation Algorithms](#)

Recommended Reading

1. Almeida V.T. and Guting R.H. Indexing the trajectories of moving objects in networks. *GeoInformatica*, 9(1):33–60, 2005.
2. Arumugam S. and Jermaine C. Closest-point-of-approach join for moving object histories. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 86.
3. Bakalov P., Hadjieleftheriou M., Keogh E., and Tsotras V. Efficient trajectory joins using symbolic representations. In Proc. 6th Int. Conf. on Mobile Data Management, 2005, pp. 86–93.
4. Brakatsoulas S., Pfoser D., Salas R., and Wenk C. On map-matching vehicle tracking data. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 853–864.
5. Chen L., Özsu M.T., and Oria V. Robust and fast similarity search for moving object trajectories, In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 491–502.
6. Chomicki J. and Revesz P. A geometric framework for specifying spatiotemporal objects. In Proc. 6th Int. Workshop Temporal Representation and Reasoning, 1999, pp. 41–46.
7. Erwig M., Güting R.H., Schneider M., and Varzigiannis M. Spatio-temporal data types: an approach to modeling and querying moving objects in databases. *GeoInformatica*, 3(3):265–291, 1999.
8. Frentzos E., Gratsias K., Pelekis N., and Theodoridis Y. Algorithms for nearest neighbor search on moving object trajectories. *Geoinformatica*, 11(2):159–193, 2007.
9. Frentzos E., Gratsias K., and Theodoridis Y. Index-based most similar trajectory search. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 816–825.
10. Forlizzi L., Güting Nardelli E., and Schneider M. A data model and data structures for moving objects databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 319–330.
11. Guting R.H., Bohlen M.H., Erwig M., Jensen C.S., Lorentzos N.A., Schneider M., and Vazirgiannis M. A foundation for representing and querying moving objects. *ACM Trans. Database Syst.*, 25(1):1–42, 2000.
12. Keogh E. Exact indexing of dynamic time warping. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 406–417.

13. Meratnia N. and By R. Spatiotemporal compression techniques for moving point objects. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 765–782.
14. Pfoser D., Jensen C.S., and Theodoridis Y. Novel approaches to the indexing of moving object trajectories. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 395–406.
15. Tao Y., Kollios G., Considine J., Li F., and Papadias D. Spatio-temporal aggregation using sketches. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 214–226.
16. Vlachos M., Kollios G., and Gunopulos D. Discovering similar multidimensional trajectories. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 673–684.

SPC Query

- [Conjunctive Query](#)

SPCU-Algebra

- [Positive Relational Algebra](#)

Specialization

- [Abstraction](#)

Specialization and Generalization

BERNHARD THALHEIM

Christian-Albrechts University Kiel, Kiel, Germany

Synonyms

[Refinement](#); [Abstraction](#); [Hierarchies](#)

Definition

Specialization and generalization are main principles of database modeling. Specialization is based on a refinement of types or classes to more specific ones. Generalization maps or groups types or classes to more abstract or combined ones. Typically, generalizations and specializations form a hierarchy of types and classes.

Key Points

Specialization introduces a new entity type by adding specific properties belonging to that type, which are different from the general properties of its more general type. Is-A associations specialize a type to a more specific one. Is-A-Role-Of associations consider a specific behavior of objects. Is-More-Specific-To associations specialize properties of objects of the more general type. The *student* type and the *customer* type are specializations of the *person* type. The *rectangle* type is specialized to the *square* type by adding restrictions. Different kinds of specialization may be distinguished: structural specialization which extends the structure, semantic specialization which strengthens type restrictions, pragmatic specialization which allows a separation of the different usage of objects in contexts, operational specialization which introduces additional operations, and hybrid specializations. Identification and other properties of objects of the special type can be inherited from the more general one. Methods applicable to objects of the more general one should be applicable to corresponding more special objects or specialized as well. Exceptions can be modeled by specializations. Specialization allows developers to avoid null values and to hide details from non-authorized users.

Generalization combines common features, attributes, or methods of types. It is based either on abstraction, on combination or on grouping. Generalization often tends to be an abstraction in which a more general type is defined by extracting common properties of one or more types while suppressing the differences between the subtypes. The subtypes can be virtually clustered by or generalized to or combined by a view to a general type. The *library's holding* type is a generalization of the *journal*, *book*, *preprint* and *PhD/Master thesis* types. The *occupation* type is a generalization of the *lawyer*, *merchant*, *teacher* and *banker* types. It is obtained by factoring out the commonalities among the specializations. Structural combination typically assumes the existence of a unifiable identification of all types. The *livestock* type combines the different types of farming. Generalization is represented by clusters of types. The cluster construct of the extended ER model represents common properties and abstractions. Identification of generalized objects is either inherited from the more special objects or built as an abstraction of the identification of the more special types. Generalizations often do not have their own methods.

Cross-references

► [Database Fundamentals – Semantic Data Models](#)

Recommended Reading

1. Ter Bekke J.H. Semantic Data Modeling, Prentice-Hall, London, 1992.
2. Thalheim B. Entity-Relationship Modeling – Foundations of Database Technology. Springer, Berlin Hiedelberg New York, 2000.

Specificity

JOVAN PEHCEVSKI¹, BENJAMIN PIWOWARSKI²

¹INRIA Paris Rocquencourt, Le Chesnay Cedex, France

²University of Glasgow, Glasgow, UK

Synonyms

[Coverage](#)

Definition

Specificity is a relevance dimension that describes the extent to which a document part focuses on the topic of request. In the context of structured text (XML) retrieval, a document part corresponds to an XML element.

Specificity is defined as the length ratio, typically in number of characters, of contained relevant to irrelevant text in the document part. Different Specificity values can be associated to a document part. These values are drawn from the Specificity relevance scale, which has evolved from a discrete multi-graded relevance scale to a continuous relevance scale.

Key Points

The Initiative for the Evaluation of XML Retrieval (INEX) has defined Specificity as a relevance dimension that uses values from its own relevance scale to express the extent to which an XML element focuses on the topic of request. Since 2002, different names and relevance scales were used for Specificity at INEX. It initially evolved because the relevance dimension was not sufficiently well defined, and later because the assessment procedure changed.

In 2002, Specificity was named coverage at INEX, which reflected the extent to which an XML element was focused on aspects of the information need (as represented by the INEX topic). The component

coverage used a relevance scale comprising four relevance grades, from “no coverage,” “too large,” “too small,” to “exact coverage.” However, this dimension was used solely in 2002, partly because of the vagueness introduced in the terminology for its name, and partly because it has been subsequently shown that the INEX 2002 assessors did not particularly understand the notion of “too small” [1]. In particular, assessors understood “too small” as a measure of quantity while Specificity is more related to the concentration of relevant information. In 2003 and 2004, four grades were used for the Specificity relevance dimension at INEX, such that the extent to which an XML element may focus on the topic of request could range from “none” (0), to “marginally” (1), to “fairly” (2), or to “highly” (3) focused. An XML element was considered relevant only if its Specificity value was greater than zero.

From 2005 onwards, a highlighting assessment procedure was used at INEX to gather relevance assessments for the XML retrieval topics. The Specificity of an XML element is automatically computed as the ratio of highlighted to fully contained text, where the relevance values that can be associated to the element are drawn from a continuous relevance scale. These values are in the range between 0 and 1, where the value of 0 corresponds to an element that does not contain any highlighted text, while the value of 1 corresponds to a fully highlighted element.

With the highlighting assessment procedure, assessors are asked to highlight all the relevant information contained by returned XML documents. This results in a reduced cognitive load on the assessor, since in this case there is no need for the assessor to explicitly associate a Specificity value to a judged element. Studies of the level of assessor agreement, which used topics that were double-judged at INEX, have shown that the use of the new highlighting procedure further increases the level of assessor agreement compared to the level of agreement observed among assessors during previous years at INEX [2,3].

Cross-references

- ▶ Evaluation Metrics for Structured Text Retrieval
- ▶ Relevance

Recommended Reading

1. Kazai G., Masood S., and Lalmas M. A study of the assessment of relevance for the INEX 2002 test collection. In Proc. 26th European Conf. on IR Research, 2004, pp. 296–310.

2. Pehcevski J. and Thom J.A. HiXEval: highlighting XML retrieval evaluation. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, pp. 43–57.
3. Trotman A. Wanted: element retrieval users. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 63–69.

Spectral Clustering

SERGIOS THEODORIDIS¹,

KONSTANTINOS KOUTROUMBAS²

¹University of Athens, Athens, Greece

²Institute for Space Applications and Remote Sensing, Athens, Greece

Synonyms

Graph-based clustering

Definition

Let X be a set $X = \{x_1, x_2, \dots, x_N\}$ of N data points. An m -clustering of X , is defined as the partition of X into m sets (*clusters*), C_1, \dots, C_m , so that the following three conditions are met:

- $C_i \neq \emptyset, i = 1, \dots, m$
- $\bigcup_{i=1}^m C_i = X$
- $C_i \cap C_j = \emptyset, i \neq j, i, j = 1, \dots, m$

In addition, the data points contained in a cluster C_i are “more similar” to each other and “less similar” to the points of the other clusters. The terms “similar” and “dissimilar” depend very much on the types of clusters the user expects to recover from X . A clustering defined as above is known as hard clustering, to distinguish it from the fuzzy clustering case.

Historical Background

The essence of clustering is to “reveal” the organization of patterns into “sensible” groups. It has been used as a critical analysis tool in a vast range of disciplines, such as medicine, social sciences, engineering, computer science, machine learning, bioinformatics, data mining and information retrieval. The literature is huge and numerous techniques have been suggested over the years. A comprehensive introduction to clustering can be found e.g., in [14]. An important class of clustering algorithms builds around graph theory. Reference [9] is one of the first efforts in this direction. Points of X are assigned to the nodes of a graph.

Notions such as minimum spanning tree and directed trees have extensively been used to partition the graph into clusters (e.g., [14]). Spectral clustering is a more recent class of graph based techniques, which unravels the structural properties of a graph using information conveyed by the spectral decomposition (eigendecomposition) of an associated matrix. The elements of this matrix code the underlying similarities among the nodes (data points) of the graph (e.g., [3]). Among the earlier works on spectral clustering are [6,12]. Fiedler [4] was one of the first to show the application of eigenvectors in graph partitioning.

Foundations

In the sequel, the simplest task of partitioning a given data set, X , into two clusters, A and B , is considered. Let $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\} \subset R^l$, where the latter denotes the l -dimensional Euclidean space. According to the previous discussion, the following preliminary steps are in order:

- Construction of a graph $G(V, E)$, where each vertex of the graph corresponds to a point \mathbf{x}_i , $i = 1, 2, \dots, N$, of X . It is further assumed that G is undirected and connected. In other words, there exists at least one path of edges that connects any pair of points in the graph.
- Assignment of a weight $W(i, j)$ to each one of the edges of the graph, e_{ij} , that quantifies proximity between the respective nodes, v_i, v_j in G (For notational convenience in some places i is used instead of v_i). The set of weights defines the $N \times N$ weight matrix W , also known as *affinity* matrix, with elements

$$W \equiv [W(i, j)], \quad i, j = 1, 2, \dots, N$$

The weight matrix is assumed to be symmetric, i.e., $W(i, j) = W(j, i)$. The choice of the weights is carried out by the user and it is a problem dependent task. A common choice is

$$W(i, j) = \begin{cases} \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right), & \text{if } \|\mathbf{x}_i - \mathbf{x}_j\| < \epsilon \\ 0, & \text{otherwise} \end{cases}$$

where ϵ is a user-defined constant and $\|\cdot\|$ is the Euclidean norm in the l -dimensional space.

By the definition of clustering, $A \cup B = X$ and $A \cap B = \emptyset$. Once a weighted graph has been formed, the second phase in any graph-based clustering algorithm consists of the following two steps: (i) Choose an

appropriate clustering criterion for the partitioning of the graph and (ii) Adopt an efficient algorithmic scheme to determine the partitioning that optimizes the previous clustering criterion.

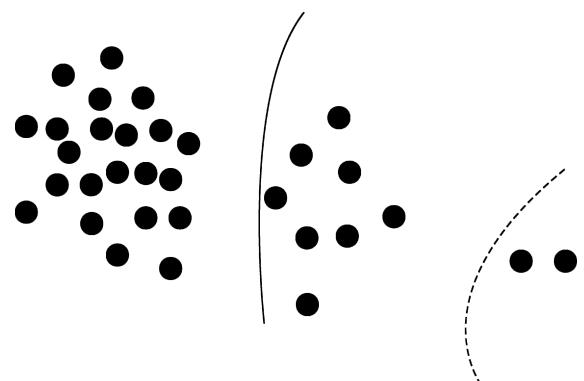
A clustering optimality criterion, that is in line with “common sense,” is the so called *cut* [16]. If A and B are the resulting clusters, the associated *cut* is defined as:

$$\text{cut}(A, B) = \sum_{i \in A, j \in B} W(i, j) \quad (1)$$

Selecting A and B so that the respective $\text{cut}(A, B)$ is minimized means that the set of edges, connecting nodes in A with nodes in B , have the minimum sum of weights, indicating the lowest similarity between points in A and B . However, this simple criterion turns out to form clusters of small size of isolated points (least similar with the rest of the nodes). This is illustrated in Fig. 1. The minimum *cut* criterion would result in the two clusters separated by the dotted line, although the partition by the full line seems to be a more natural partitioning.

To overcome this drawback, the *normalized cut* criterion has been suggested in [12]. This is one of the most commonly used criteria in spectral clustering. The essence of this criterion is to minimize the *cut* and at the same time trying to keep the sizes of the formed clusters large. To this end, for each node, $v_i \in V$, in the graph G the index

$$D_{ii} = \sum_{j \in V} W(i, j) \quad (2)$$



Spectral Clustering. Figure 1. The dotted line indicates the clustering that is likely to favor the minimum *cut* criterion, while the full line indicates a more natural partitioning.

is defined. This is an index indicative of the “importance” of a node, v_i , $i = 1, 2, \dots, N$. The higher the value of D_{ii} the more similar the i th node is to the rest of the nodes. A low D_{ii} value indicates an isolated (remote) point. Given a cluster A , a measure of the “importance” of A is given by the following index

$$V(A) = \sum_{i \in A} D_{ii} = \sum_{i \in A, j \in V} W(i, j) \quad (3)$$

where $V(A)$ is sometimes known as the *volume* or the *degree* of A . It is obvious that small and isolated clusters will have a small $V(\cdot)$. The *normalized cut* between two clusters A, B is defined as

$$Ncut(A, B) = \frac{cut(A, B)}{V(A)} + \frac{cut(A, B)}{V(B)} \quad (4)$$

Obviously, small clusters correspond to large values (close to one) for the previous ratios, since in such cases $cut(A, B)$ will be a large percentage of $V(A)$.

Minimization of the $Ncut(A, B)$ turns out to be an NP-hard task. To bypass this computational obstacle, the problem will be reshaped to a form that allows an efficient approximate solution. To this end let ([1])

$$\gamma_i = \begin{cases} \frac{1}{V(A)}, & \text{if } i \in A \\ -\frac{1}{V(B)}, & \text{if } i \in B \end{cases} \quad (5)$$

$$\mathbf{y} = [y_1, y_2, \dots, y_N]^T$$

where T denotes the transpose operation. After some algebraic manipulations it can be verified that

$$\left(\frac{1}{V(A)} + \frac{1}{V(B)} \right)^2 cut(A, B) \propto \frac{1}{2} \sum_{i \in V} \sum_{j \in V} (y_i - y_j)^2 W(i, j) = \mathbf{y}^T L \mathbf{y} \quad (6)$$

where \propto denotes proportionality and

$$L = D - W, \quad D \equiv \text{diag}\{D_{ii}\}$$

is known as the *graph Laplacian* matrix and D is the diagonal matrix having the elements D_{ii} across the main diagonal. It is also easily verified that

$$\mathbf{y}^T D \mathbf{y} = \frac{1}{V(A)} + \frac{1}{V(B)} \quad (7)$$

Combining (4), (6) and (7) it turns out that minimizing $Ncut(A, B)$ is equivalent with minimizing

$$J = \frac{\mathbf{y}^T L \mathbf{y}}{\mathbf{y}^T D \mathbf{y}} \quad (8)$$

subject to the constraint that $y_i \in \{\frac{1}{V(A)}, -\frac{1}{V(B)}\}$. Furthermore, based on the respective definitions, it can be shown that

$$\mathbf{y}^T D \mathbf{1} = 0 \quad (9)$$

where $\mathbf{1}$ is the N -dimensional vector having all its elements equal to 1. In order to bypass the computationally hard nature of the original task a relaxed problem will be solved: Eq. (8) will be minimized subject to the constraint of (9). The unknown “cluster labels,” y_i , $i = 1, 2, \dots, N$, are now allowed to move freely along the real axis. Let

$$\mathbf{z} \equiv D^{1/2} \mathbf{y}$$

Then (8) becomes

$$J = \frac{\mathbf{z}^T \tilde{L} \mathbf{z}}{\mathbf{z}^T \mathbf{z}} \quad (10)$$

and the constraint in (9)

$$\mathbf{z}^T D^{1/2} \mathbf{1} = 0 \quad (11)$$

Matrix $\tilde{L} \equiv D^{-1/2} L D^{-1/2}$ is known as the *normalized graph Laplacian* matrix. It can easily be shown that \tilde{L} has the following properties

- It is symmetric, real valued and nonnegative definite. Thus, as it is known from linear algebra, all its eigenvalues are non-negative and the corresponding eigenvectors are orthogonal to each other.
- $D^{1/2} \mathbf{1}$ is an eigenvector corresponding to zero eigenvalue. Indeed,

$$\tilde{L} D^{1/2} \mathbf{1} = 0$$

Obviously $\lambda = 0$ is the smallest eigenvalue of \tilde{L} , due to the non-negative definite nature of the matrix.

Furthermore, the ratio in (10) is the celebrated Rayleigh quotient for which the following hold (e.g., [5])

- The smallest value of the quotient, with respect to \mathbf{z} , is equal to the smallest eigenvalue of \tilde{L} and it occurs for \mathbf{z} equal to the eigenvector corresponding to this (smallest) eigenvalue.
- If the solution is constrained to be orthogonal to all eigenvectors associated with the j smaller

eigenvalues, minimization of the Rayleigh quotient results to the eigenvector corresponding to the next smallest eigenvalue, λ_{j+1} and the minimum value is equal to λ_{j+1} .

Taking into account i) the orthogonality condition in the constraint (11) and ii) the fact that $D^{1/2}\mathbf{1}$ is the eigenvector corresponding to the smallest eigenvalue $\lambda_0 = 0$ of \tilde{L} , it follows that:

The optimal solution vector \mathbf{z} minimizing the Rayleigh quotient in (10), subject to the constraint (11), is the eigenvector corresponding to the second smallest eigenvalue of \tilde{L} .

In summary, the basic steps of the spectral clustering algorithm are:

- Given a set of points $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ the weighted graph $G(V, E)$ is constructed. Then, the weight matrix W is formed by adopting a similarity rule.
- The matrices D , $L = D - W$ and \tilde{L} are formed. The eigenanalysis of the normalized Laplacian matrix

$$\tilde{L}\mathbf{z} = \lambda\mathbf{z}$$

is performed and the computation of the eigenvector \mathbf{z}_1 corresponding to the second smallest eigenvalue λ_1 of \tilde{L} is carried out. Then $\mathbf{y} = D^{-1/2}\mathbf{z}_1$ is computed.

- Finally, discretization of the components of \mathbf{y} according to a threshold value takes place.

The final step is necessary since the components of the obtained solution are real-valued and our required solution is binary. To this goal, different techniques can be applied. For example, the threshold can be taken to be equal to zero. Another choice is to adopt the median value of the components of the optimum eigenvector. An alternative approach would be to select the threshold value that gives the minimum *cut*.

The eigenanalysis or spectral decomposition, as it is sometimes called, of an $N \times N$ matrix, using a general purpose solver, amounts to $O(N^3)$ operations. Thus, for large number of data points, this may be prohibitive in practice. However, for most of the practical applications the resulting graph is only locally connected, and the associated affinity matrix is a *sparse* one. Moreover, only the smallest eigenvalues/eigenvectors are required and also the accuracy is not of major issue, since the solution is to be discretized. In such a setting, the efficient Lanczos algorithm can be mobilized and the computational requirements drop down to approximately $O(N^{3/2})$.

So far, the partition of a data set into two clusters has been considered. If more clusters are expected, the scheme can be used in a hierarchical mode, where, at each step, each one of the resulting clusters is divided into two partitions. This is continued until a prespecified criterion is satisfied.

In the discussion above the focus was on a specific clustering criterion, i.e., the normalized cut, in order to present the basic philosophy behind the spectral clustering techniques. No doubt, a number of other criteria have been proposed in the related literature, e.g., [8,13]. In [15] a review and a comparative study of a number of popular spectral clustering algorithms is presented.

Key Applications

Spectral clustering has been used in a number of applications such as image segmentation and motion tracking [13,11], circuit layout [2], gene expression [8], machine learning [10], load balancing [7].

Cross-references

- ▶ [Clustering Overview and Applications](#)
- ▶ [Graph](#)
- ▶ [Hierarchical Clustering](#)
- ▶ [Image Segmentation](#)

Recommended Reading

1. Belkin M. and Niyogi P. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Comput.*, 15(6):1373–1396, 2003.
2. Chan P., Schlag M., and Zien J. Spectral k -way ratio cut partitioning. *IEEE Trans. Comput. Aided Des. Integrated Circ. Syst.*, 13:1088–1096, 1994.
3. Chung F.R.K. *Spectral Graph Theory*. American Mathematical Society, 1997.
4. Fiedler M. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory. *Czechoslovak Math. J.*, 25(100):619–633, 1975.
5. Golub G.H. and Van Loan C.F. *Matrix Computations*. John Hopkins, Baltimore, MD, USA, 1989.
6. Hagen L.W. and Kahng A.B. New spectral methods for ratio cut partitioning and clustering. *IEEE Trans. Comput. Aided Des. Integrated Circ. Syst.*, 11(9):1074–1085, 1992.
7. Hendrickson B. and Leland R. Multidimensional spectral load balancing. In Proc. Fourth SIAM Conf. on Parallel Processing. 1993, pp. 953–961.
8. Kannan R., Vempala S., and Vetta A. On clusterings- good, bad and spectral. In Proc. 41st Annual Symp. on Foundations of Computer Science, 2000, pp. 367–377.
9. Ling R.F. On the theory and construction of k -clusters. *Comput. J.*, 15:326–332, 1972.

10. Ng A.Y., Jordan M., and Weiss Y. On spectral clustering analysis and an algorithm. In Proc. 14th Conf. on Advances in Neural Information Processing Systems, 2001.
11. Qiu H. and Hancock E.R. Clustering and embedding using commute times. IEEE Trans. Pattern Anal. Mach. Intell., 29(11):1873–1890, 2007.
12. Scott G. and Longuet-Higgins H. Feature grouping by relocalization of eigenvectors of the proximity matrix. In Proc. British Machine Vision Conf., 1990, pp. 103–108.
13. Shi J. and Malik J. Normalized cuts and image segmentation. IEEE Trans. Pattern Anal. Mach. Intell., 22(8):888–905, 2000.
14. Theodoridis S. and Koutroumbas K. Pattern Recognition (4th edn.). Academic Press, 2008.
15. Verma D. and Meilă M. A comparison of spectral clustering algorithms. Technical report, UW-CSE-03-05-01, CSE Department, University of Washington, Seattle, 2003.
16. Wu Z. and Leahy R. An optimal graph theoretic approach to data clustering: Theory and its applications to image segmentation. IEEE Trans. Pattern Anal. Mach. Intell., 15(11):1101–1113, 1993.

Spider

- Web Crawler Architecture

Spidering

- Incremental Crawling

SPJRU-Algebra

- Positive Relational Algebra

Split

NATHANIEL PALMER

Workflow Management Coalition, Hingham,
MA, USA

Synonyms

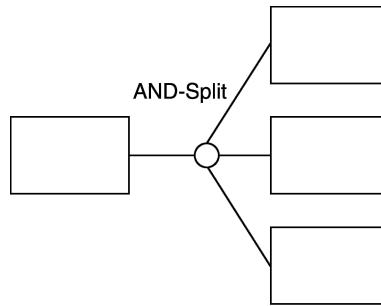
- AND-split

Definition

A point within the workflow where a single thread of control splits into two or more parallel activities.

Key Points

The execution of parallel activities commences with an AND-Split and concludes with an AND-Join. For example, in a credit application process there may be a split in the workflow at which point multiple activities are completed separately (in parallel, if not simultaneously.) At an And-Split separate threads of control within the process instance are created; these threads will proceed autonomously and independently until reaching an And-Join condition. In certain workflow systems, all the threads created at an And-Split must converge at a common And-Join point (Block Structure);



in other systems convergence of a subset of the threads can occur at different And-Join points, potentially including other incoming threads created from other And-split points.

Cross-references

- AND-Join
- OR-Split

Split Transactions

GEORGE KARABATIS

University of Maryland, Baltimore County (UMBC),
Baltimore, MD, USA

Definition

The split-transaction is an extended transaction model that introduces two new transaction management primitives/operations, namely, split and join. The split operation on a transaction T splits T and replaces it with two serialisable transactions; each one is later committed or aborted independently of the other. The inverse of split is the join operation on a transaction T

which dissolves T by joining its results with a target transaction S.

Key Points

The concept of split transactions was introduced by Pu, Kaiser, and Hutchinson in [3] and later elaborated in [2] to support open-ended activities such as CAD/CAM projects, engineering type of applications, and software development. The syntax of the split-transaction operation on transaction T produces two new transactions A and B and dissolves T [3,2]:

```
Split-Transaction (
  A: (AReadSet, AWriteSet, AProcedure),
  B: (BReadSet, BWriteSet, BProcedure))
```

where AReadSet, AWriteSet, BReadSet, BWriteSet are sets of data items accessed by A and B. AProcedure and BProcedure are the starting points of code where A and B will begin execution. There is no need to explicitly mention T in the arguments of the operation, as by definition the operation can only be executed within the body of transaction T. The syntax for a join operation on T to join S is: Join-Transaction (S: TID)

The split-transaction operation can be used to commit some work of a transaction early, or to distribute on-going work among several coworkers. On the contrary, the join-transaction is used to hand over and integrate results with a coworker [2]. These operations pertain to programmed transactions, and to actual open-ended activities with unpredictable developments (users determine the next operation to be executed in an adhoc manner). In the latter case, the read sets and write sets defined above are replaced with data sets on which application specific operations (e.g., edit or compile for software development) are applied; the AProcedure and Bprocedure are replaced with AUser and BUser specifying the users who take control of A and B. Thus, the definition changes to:

```
Split-Transaction (
  A: (AReadSet, AWriteSet, AUser),
  B: (BReadSet, BWriteSet, BUser))
```

There are additional operations such as: Split-Commit (transaction A is immediately committed, while B may be taken over by BUser; also Suspend (giving up control of a transaction) and Accept-Join-Transaction (a user executes this operation to accept responsibility of a joined transaction).

The main advantages of the restructuring operations (split/join transaction) on open-ended activities are:

- Adaptive recovery: committing resources that will not change
- Added concurrency: releasing committed resources or transferring ownership of uncommitted resources
- Serializable access to resources by all activities

The split and join primitives were subsequently incorporated with nested transactions to produce combined transaction models [1].

Cross-references

- ▶ [Database Management System](#)
- ▶ [Distributed Transaction Management](#)
- ▶ [Extended Transaction Models](#)
- ▶ [Serializability](#)
- ▶ [Transaction](#)
- ▶ [Transaction Management](#)
- ▶ [Transaction Manager](#)

Recommended Reading

1. Chrysanthis P.K. and Ramamritham K. Synthesis of extended transaction models using ACTA. ACM Trans. Database Syst., 19 (3):450–491, 1994.
2. Kaiser G.E. and Pu C. Dynamic restructuring of transactions. In Database Transaction Models for Advanced Applications, A. K. Elmagarmid (ed.). Morgan Kaufmann Publishers, 1992, pp. 265–295.
3. Pu C., Kaiser G.E., and Hutchinson N.C. Split-transactions for open-ended activities. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988, pp. 26–37.

SQL

DON CHAMBERLIN

IBM Almaden Research Center, San Jose, CA, USA

Synonyms

[SEQUl](#); Structured Query Language

Definition

SQL is the world's most widely-used database query language. It was developed at IBM Research Laboratories in the 1970s, based on the relational data model defined by E. F. Codd in 1970. It supports retrieval, manipulation, and administration of data stored in tabular form. It is the subject of an international standard named Database Language SQL.

Historical Background

Early Language Development

In June 1970, E. F. Codd of IBM Research published a paper [4] defining the relational data model and introducing the concept of data independence. Codd's thesis was that queries should be expressed in terms of high-level, nonprocedural concepts that are independent of physical representation. Selection of an algorithm for processing a given query could then be done by an optimizing compiler, based on the access paths available and the statistics of the stored data; if these access paths or statistics should later change, the algorithm could be re-optimized without human intervention. In a series of papers, Codd proposed two high-level languages for querying relational databases, called relational algebra and relational calculus (also known as Data Sublanguage Alpha) [5].

The advantages of the relational model for application developers and database administrators were immediately clear. It was less clear whether an optimizing compiler could consistently translate nonprocedural queries into algorithms that were efficient enough for use in a production database environment. To investigate this issue, IBM convened a project called System R [1] at its research laboratory in San Jose, California. Between 1973 and 1979, this project designed and implemented a prototype relational database system based on Codd's ideas, testing and refining the prototype in several customer locations. SQL was the user interface defined by the System R research project. It later became the user interface for relational database products marketed by IBM and several other companies.

The principal goals that influenced the design of SQL were as follows:

1. SQL is a high-level, nonprocedural language intended for processing by an optimizing compiler. It is designed to be equivalent in expressive power to the relational query languages originally proposed by Codd.
2. SQL is intended to be accessible to users without formal training in mathematics or computer programming. It is designed to be typed on a keyboard. Therefore it is framed in familiar English keywords, and avoids specialized mathematical concepts or symbols.
3. SQL attempts to unify data query and update with database administration tasks such as creating and

modifying tables and views, controlling access to data, and defining constraints to protect database integrity. In pre-relational database systems, these tasks were usually performed by specialized database administrators and required shutting down and reconfiguring the database. By building administrative functions into the query language, SQL helps to eliminate the database administrator as a choke point in application development.

4. SQL is designed for use in both decision support and online transaction processing environments. The former environment requires processing of complex queries, usually executed infrequently but accessing large amounts of data. The latter environment requires high-performance execution of parameterized transactions, repeated frequently but accessing (and often updating) small amounts of data. Both end-user interfaces and application programming interfaces are necessary to support this spectrum of usage.

The first specification of SQL was published in May 1974, in a 16-page conference paper [3] by Don Chamberlin and Ray Boyce, members of the System R project. In this paper, the language was named SEQUEL, an acronym for Structured English Query Language. The paper included a BNF syntax for the proposed language. This original paper presented only basic query features, without any facilities for data definition or update. However, the basic structure of the language, including query-blocks, grouping, set operations, and aggregating functions, has been consistent from this paper to the present day.

Over the course of the System R project, SEQUEL continued to evolve based on experience gathered by users and implementers. A much more complete description of the language was published in the IBM Journal of Research and Development in November 1976 [2], including data manipulation facilities (insert, delete, and update), a more complete join facility, facilities for defining tables and views, and database administration facilities including access control, assertions, and triggers. In 1977, because of a trademark issue, the name SEQUEL was shortened to SQL.

Although SQL was designed and prototyped at IBM Research, the language was published in the open literature, and the first commercial SQL product was released by a small company called Relational Software, Inc., in 1979. This product was named

Oracle, a name that was later adopted by the company, which is no longer small. The first IBM product based on SQL was called SQL/Data System, released in 1981, followed by DB2, released in 1983 on mainframes and eventually supported on many IBM platforms. SQL has now been implemented by all major database vendors and is available in a wide variety of operating environments. In addition to commercial database products, SQL implementations include several popular open-source products such as MySQL (<http://www.mysql.com>) and Apache Derby (<http://db.apache.org/derby/>).

Standards

Shortly after the first appearance of SQL in a commercial product, an effort was made to standardize the language. Over the years, the SQL standard has contributed to the growth of the database industry by defining a common interface for use by database vendors, application developers, and tools.

The first SQL standard, named “Database Language SQL,” was published by the American National Standards Institute (ANSI) in 1986 (Standard No. X3.135-1986), and an identical standard with the same name was published by the International Standards Organization (ISO) in 1987 (Standard No. ISO 9075-1987). Over the years, ANSI and ISO have cooperated to keep their respective SQL standards synchronized as they have evolved through several versions.

The original standard, often called SQL-86, occupied just under 100 pages and included only simple queries, updates, and table definitions. It was followed in 1989 by a revised standard that added several kinds of constraints for protecting the integrity of stored data. This version of the standard comprised about 120 pages and is often referred to as SQL-89 (“Database Language SQL with Integrity Enhancement”).

A major revision of the SQL standard, usually called SQL-92, was published by ANSI and ISO/IEC in 1992. This version improved the orthogonality of the language, allowing expressions to be used wherever tables or scalar values are expected. SQL-92 also added several new features, including date and time data-types, set-oriented operators such as UNION and INTERSECT, standard catalog tables for storing metadata, and schema-evolution features such as ALTER TABLE. SQL-92 comprised about 600 pages.

A conformance test suite for SQL-92 was developed in the United States by the National Institute of Standards and Technology (NIST). After certifying several conforming products, NIST discontinued SQL conformance testing in 1996.

Between 1992 and 1999, two specialized extensions to SQL-92 were published, named Call Level Interface (CLI) and Persistent Stored Modules (PSM). CLI defines a set of functions whereby programs written in languages such as C can dynamically connect to relational databases and execute SQL statements. PSM extends SQL with assignment statements, control-flow statements, and exception handlers, making it possible to implement some database applications entirely in SQL. PSM was an attempt to standardize the procedural extensions such as PL/SQL [9] and Transact-SQL [12] that had been added to SQL by several database vendors.

The next major update of the SQL standard occurred in 1999 and is usually called SQL:1999. This new version split the standard into several parts, incorporating CLI as Part 3 and PSM as Part 4. SQL:1999 introduced important new functionality including triggers, large objects, recursive queries, and user-defined functions. It placed major emphasis on object-relational functionality and on new features for on-line analytic processing (OLAP). Details of these and other recent additions to SQL are described below under “Advanced Features.” The sum of all the parts of SQL:1999 exceeded 2,000 pages. Additional parts continue to be added to the SQL standard from time to time. The most recent major revision of the standard, known as SQL:2003, with all its parts, comprises more than 3,600 pages.

Over the years, the SQL standard has provided a controlled framework within which the language can evolve to correct its initial limitations and to meet changing user requirements. The standard has also served to focus the industry’s attention and resources, providing a common framework in which individuals and companies could develop tools, write books, teach courses, and provide consulting services. The standard has been only partially successful in making SQL applications portable across implementations; this goal has been hampered by the fact that different vendors have implemented different subsets of the standard, and by the lack (since 1996) of a test suite to validate conformance of an implementation. The latest versions of the various parts of the standard can be obtained from ISO

[7] or from national standards organizations such as ANSI [8]. Jim Melton, editor of the SQL standard, has also published a two-volume reference book explaining the standard in a very accessible style [11, see also 10].

Foundations

Queries

SQL operates on data in the form of tables. Each table has a name and consists of one or more columns, each of which has a name and a datatype. The content of a table consists of zero or more rows, each of which has a value for each of the columns. The value associated with a given row and column may be an instance of the datatype of that column, or may be a special “null” value indicating that the value is missing (not available or not applicable). SQL statements, which may be queries or updates, operate on stored tables or on tabular “views” that are derived from stored tables. The result of a query is an unnamed virtual table. The result of an update is a change to the stored data, which is visible to subsequent statements. Generally, updates can be applied to a view only if each row in the view can be mapped uniquely onto a row of a stored table. This rule makes it possible to map updates on the view to updates on the underlying table.

An SQL query consists of one or more *query-blocks*. A query-block consists of several clauses, each of which begins with a keyword. Some of these clauses (keywords SELECT and FROM) are required, and others (keywords WHERE, GROUP BY, and HAVING) are optional. The examples in this article use upper-case keywords and lower-case names, although SQL is a case-insensitive language. The examples are based on two tables named PARTS and SUPPLIERS. The primary key of PARTS is PARTNO and the primary key of SUPPLIERS is SUPPNO. SUPPNO also appears as a foreign key in the PARTS table (see “Database Administration” below for definitions of primary key and foreign key.)

The following query-block illustrates a join of two tables. Conceptually, rows from the PARTS table are paired with rows from the SUPPLIERS table according to the criterion specified in the WHERE clause (SUPPNO’s must match), and the resulting row-pairs are filtered by an additional condition (supplier’s location must be Denver). From the surviving row-pairs, the SELECT clause specifies the columns that appear in the query result (in this case, the part number and the supplier name). Many different strategies are possible

for executing this query; since SQL is a non-procedural language, choice of an execution strategy is left to an optimizing compiler.

```
SELECT p.partno, s.name
FROM parts p, suppliers s
WHERE p.suppno = s.suppno
AND s.location = 'Denver'
```

The following query-block illustrates grouping and aggregation. Conceptually, the rows of the PARTS table are partitioned into groups with matching SUPPNO values. The groups are then filtered by the HAVING clause, which applies a predicate based on group properties (in this case, retaining only groups that have at least ten rows.) Finally, the SELECT clause specifies the columns of the query result, which consists of one row for each group. In a grouping query, the selected expressions must consist of group properties (grouping keys or aggregating functions such as avg, sum, max, min, and count.) The SELECT clause can also specify names for the output columns. The query-block in this example returns, for each supplier that supplies at least ten parts, the average cost of all parts supplied by that supplier.

```
SELECT suppno, avg(cost) AS avgcost
FROM parts
GROUP BY suppno
HAVING count(*) >= 10
```

Many queries, including the above examples, consist of a single query-block. Query-blocks can also be combined to form larger queries. In the following example, a query-block serves as a subquery that computes a value used in another query-block. The subquery finds the maximum cost in the PARTS table, and the outer query-block returns the part numbers of the parts that have this maximum cost. The query-blocks are linked together by the IN keyword, denoting that the value in the WHERE-clause is tested for inclusion in the list of values returned by the subquery, which may in general return multiple values.

```
SELECT partno
FROM parts
WHERE cost IN
      (SELECT max(cost)
       FROM parts)
```

Another way in which query-blocks can be combined to form a larger query is by means of the set-operators UNION, INTERSECT, and EXCEPT, which

compute the union, intersection, or difference of the sets of rows returned by two query-blocks (after eliminating duplicate rows). The datatypes of the rows returned by the respective query-blocks must be compatible. The following example computes the difference of two query-blocks to find the suppliers that supply no parts. It also includes an ORDER BY clause, which specifies an ordering for the rows in the query result. The ORDER BY clause applies to the whole query rather than to an individual query-block.

```
SELECT suppno
  FROM suppliers
EXCEPT
  SELECT suppno
    FROM parts
   ORDER BY suppno
```

The first example above showed how the condition for joining two tables can be specified in a WHERE-clause. This method returns only data for which a matching row exists in both tables. Another join method called an *outer join* can be used to include rows from one of the joined tables that have no matching rows in the other table. The following example returns the supplier numbers of suppliers located in Denver, together with the parts that they supply, including Denver suppliers that supply no parts (the latter suppliers will appear in the query result with a null value in the PARTNO column):

```
SELECT s.suppno, p.partno
  FROM suppliers s LEFT OUTER JOIN parts p
    ON s.partno = p.partno
   AND s.location = 'Denver'
```

Data Manipulation

The data manipulation facilities of SQL consist of the INSERT, DELETE, and UPDATE statements, which are used for inserting rows into a table, deleting rows from a table, and updating the values of rows in a table. The values to be used in data manipulation statements may be specified as constants or computed by subqueries, as illustrated by the following examples.

This example updates the PARTS table, increasing the cost of all the parts supplied by a certain supplier by 10%:

```
UPDATE parts
SET cost = cost * 1.1
WHERE suppno = '105'
```

The following example deletes from the SUPPLIERS table all the suppliers that supply no parts. This example illustrates a subquery that is linked to the outer DELETE statement by the keywords NOT EXISTS. Notice that the subquery depends on a value in a row supplied by the outer UPDATE statement (referred to as s.supplier); therefore the subquery must be executed repeatedly for each row in the table being updated. This kind of subquery is called a *correlated subquery*.

```
DELETE FROM suppliers s
 WHERE NOT EXISTS
   (SELECT partno
     FROM parts
    WHERE suppno = s.supplier)
```

Database Administration

In addition to queries and data manipulation, SQL provides facilities for performing database administration tasks. These tasks fall into three general categories: data definition, access control, and active data features.

SQL data definition facilities include statements for creating or dropping tables and views, and for adding or deleting columns of existing tables. The definition of a view takes the form of an SQL query specifying how the view can be derived from stored tables and/or other views.

SQL access control facilities are based on the concepts of users, privileges, and roles. A privilege is the ability to perform an action (such as select, insert, delete, or update) on an object (such as a table or, in some cases, a column of a table). A role is a set of privileges that can be granted to a set of users. In general, the user who creates an object can grant privileges on that object to individual users and to roles, and can also revoke these privileges. When granting a privilege, the grantor can specify whether the grantee is authorized to pass along the privilege to additional users or roles.

SQL active data facilities include constraints and triggers. Constraints serve to enforce database integrity by limiting the kinds of updates that can be applied to a table. Important kinds of constraints include the following:

- NOT NULL constraints, which prohibit null values in a specific column.
- CHECK constraints, which specify a predicate that must not be false for any row of the table.

- PRIMARY KEY constraints, which require that the values in a specific set of columns called the *primary key* uniquely identify a row of the table.
- FOREIGN KEY constraints, which require that, for each row of the constrained table, a specific set of columns called the *foreign key* contain only combinations of values that are also found in the primary key of a related table called the *parent table*. The definer of a FOREIGN KEY constraint can specify what happens when a data manipulation statement attempts to violate the constraint (for example, the violating statement might have no effect; or deletion of a row from the parent table might cause the automatic deletion of related rows in the constrained table). Enforcement of foreign key constraints is said to protect the *referential integrity* of stored data.

A trigger is an action that is automatically invoked whenever a specific event, called the triggering event, occurs. The definer of a trigger specifies the following properties:

- The triggering event, which may be insert, delete, or update of rows (or, in some cases, columns) of a specific table.
- Whether the trigger is invoked before or after the triggering event becomes effective.
- If the triggering event affects multiple rows, whether the trigger is invoked once for each affected row or only once for the whole triggering event.
- An optional trigger condition: a predicate that must be true at the time of the triggering event in order for the trigger to be activated.
- The trigger body: one or more SQL statements that are automatically executed when the triggering event occurs and the trigger condition is true. The trigger condition and the trigger body have access to special variables that contain the data values before and after the triggering event.

Constraints and triggers are useful for specifying and enforcing the semantics of stored data. In general, it is preferable to specify a given semantic rule by means of a constraint rather than a trigger if possible, since constraints apply to all kinds of actions and provide maximum opportunities for optimization. On the other hand, some kinds of semantic rules (for example, “salaries never decrease”) can only be specified by using triggers.

Advanced Features

Over the years, a great deal of functionality has been added to the SQL language. The full set of SQL features is far too large and complex to be explained here. The following are some of the major areas in which advanced functionality has been added to SQL:

- **Recursion:** A recursive query consists of an initial subquery that computes some preliminary results and a recursive subquery that computes additional results based on values that were previously computed. The recursive subquery is executed repeatedly until no additional results are computed. Recursion is useful in queries that search some space for an optimum result, such as “Find the cheapest combination of flight segments to travel from Shanghai to Copenhagen.” Recursive queries were first defined in SQL:1999.
- **OLAP:** Online analytic processing (OLAP) is used by businesses to analyze large volumes of data to identify facts and trends that may affect business decisions. The GROUP BY clause and aggregating functions (sum, avg, etc.) of early SQL provided a primitive form of OLAP functionality, which was greatly extended in later versions of the language. For example, the ROLLUP facility enables a query to apply aggregating functions at multiple levels (such as city, county, and state). The CUBE facility enables data to be aggregated along multiple dimensions (such as date, location, and category) within a single query. The WINDOW facility allows aggregating functions to be applied to a “moving window” as it passes over a collection of data. These facilities, and others, were introduced by SQL:1999 and enhanced in subsequent versions of the standard.
- **Functions and procedures:** Originally, SQL supported a fixed collection of functions, which grew slowly over the years. SQL:1999 introduced a capability for users to define additional functions and procedures that can be invoked from SQL statements. (In this context, a procedure is simply a function that is invoked by a CALL statement and that is not required to return a value.) The bodies of user-defined functions and procedures can be written either in SQL itself or in a host language such as C or Java.
- **Object-relational features:** Early versions of SQL could process data conforming to a fixed set of

simple datatypes such as integers and strings. Over the years, a few additional datatypes such as dates and “large objects” were added. In the late 1990’s, requirements arose for a more extensible type system. SQL:1999 introduced facilities for user-defined structured types and methods. These facilities support limited forms of object-oriented functionality, including inheritance and polymorphism.

- **Multi-media:** In 2000, the SQL Standard was augmented by a separate but closely-related standard called “SQL Multimedia and Application Packages” (ISO/IEC 13249:2000), often referred to as SQL/MM. This new standard used the object-relational features introduced by SQL:1999 to define specialized datatypes and methods for text, images, and spatial data.
- **XML-related features:** XML is an increasingly popular format for data exchange because it mixes metadata (tags) with data, making the data self-describing. The popularity of XML has led to requirements to store XML data in relational databases and to convert data between relational and XML formats. These requirements have been addressed by a facility called SQL/XML, which was introduced as Part 14 of SQL:2003 and was updated in 2006. SQL/XML includes a new XML datatype, a set of functions for converting query results into XML format, and a feature whereby SQL can invoke XQuery as a sublanguage for processing stored XML data.

Criticisms

Like most widely-used programming interfaces, SQL has attracted its share of criticism. Issues that have been raised about the design of SQL include the following:

- The earliest versions of SQL lacked support for some important aspects of Codd’s relational data model such as primary keys and referential integrity. These concepts were added to the language in SQL-89, along with other integrity-related features such as unique constraints and check-constraints.
- The earliest versions of SQL had some ad-hoc rules about how various language features could be combined, and lacked the closure property because the columns of query results did not always have names. These problems were largely corrected by SQL-92.

- Null values are a complex and controversial subject. One of Codd’s famous “twelve rules” requires relational database systems to support a null value, defined as a representation of missing or inapplicable information that is systematic and distinct from all regular values [6]. Some writers believe that the complexity introduced by null values outweighs their benefit. However, there seems to be no method for dealing with missing data that is free of disadvantages. The SQL approach to this issue has been to support a null value and to allow database designers to specify, on a column-by-column basis, where nulls are permitted. One benefit of this approach has been that null values have proven useful in the design of various language features, such as outer join, CUBE, and ROLLUP, that have been added during the evolution of SQL.
- Unlike Codd’s definition of the relational data model, SQL permits duplicate rows to exist, either in a database table or in the result of a query. SQL also allows users to selectively prohibit duplicate rows in a table or in a query result. The intent of this approach is to give users control over the potentially expensive process of duplicate elimination. In some applications, duplicate rows may be meaningful (for example, in a point-of-sale system, a customer may purchase several identical items in the same transaction.) As in the case of nulls, the SQL approach has been to provide users with tools to allow or disallow duplicate rows according to the needs of specific applications.
- Another source of criticism has been the “impedance mismatch” between SQL and the host languages such as C and Java in which it is often embedded. Exchanging data between two languages with different type systems makes applications more complex and interferes with global optimization. One approach to this problem has been the development of computationally complete SQL-based scripting languages such as PSM.

Key Applications

SQL is designed to be used in a variety of application environments.

Most SQL implementations support an interactive interface whereby users can compose and execute ad-hoc SQL statements. In many cases, a graphical user interface is provided to display menus of available tables and columns and help the user to construct valid

statements. These systems also usually support menu-based interfaces for administrative functions such as creating and dropping tables and views.

More complex applications usually involve use of both SQL and a host programming language. This requires a mapping between the type systems of SQL and the host language, and a well-defined interface for exchanging data between the two environments. Interfaces have been defined between SQL and C, Java, and many other host languages. These interfaces fall into two major categories:

- **Embedded SQL:** In this approach, SQL statements are embedded syntactically in the host program, and are processed by a precompiler that extracts the SQL statements and converts them to an optimized execution plan that is invoked when the host program is executed.
- **Call interfaces:** In this approach, the host program uses function calls to establish a connection to a database and to pass SQL statements to the database system for execution. Conversion of the SQL statements to optimized execution plans is done at runtime. The most widely-used interfaces of this type are ODBC (Open Database Connectivity) [13] and JDBC (Java Database Connectivity) [14]. Since ODBC and JDBC drivers exist for many popular relational database systems, these interfaces are widely used by applications that need to access remote databases or need to be compatible with database systems from multiple vendors.

Modern web-based database applications often employ a three-tiered architecture. The *client tier* handles user interaction and communicates with a remote server via a protocol such as HTTP. Application logic resides on this server, called the *mid-tier*. The application logic, in turn, accesses an SQL database using ODBC or JDBC. The database may be implemented on the mid-tier or on a separate machine called the *database tier* or “back-end.”

Cross-references

- ▶ Computationally Complete Relational Query Languages
- ▶ Database Trigger
- ▶ Expressive Power of Query Languages
- ▶ Java Database Connectivity
- ▶ Join
- ▶ Key

- ▶ Metadata
- ▶ Multi-Tier Architecture
- ▶ Null Values
- ▶ On-Line Analytical Processing
- ▶ Open Database Connectivity
- ▶ Query Language
- ▶ Query Optimization
- ▶ Query Processing
- ▶ Relational Algebra
- ▶ Relational Calculus
- ▶ Relational Integrity Constraints
- ▶ Relational Model
- ▶ SQL Isolation Levels
- ▶ SQUARE
- ▶ Transaction
- ▶ Views
- ▶ XPath/XQuery

Recommended Reading

1. Astrahan M.M. et al. System R: A relational approach to database management. ACM Trans. Database Syst., 1(2):97–137, 1976.
2. Chamberlin D. et al. SEQUEL 2: A unified approach to data definition, data manipulation, and control. IBM J. Res. Develop., 20(6):560–575, 1976.
3. Chamberlin D. and Boyce R. SEQUEL: A structured english query language. In Proc. ACM SIGFIDET Workshop, 1974, pp. 249–264.
4. Codd E.F. A relational model of data for large shared databanks. Commun. ACM, 13(6):377–387, 1970.
5. Codd E.F. A data base sublanguage founded on the relational calculus. In Proc. ACM SIGFIDET Workshop, 1971, pp. 35–68.
6. Codd E.F. Does Your DBMS Run by the Rules? Computer-World, 21 Oct., 1985. See also http://en.wikipedia.org/wiki/Codd's_12_rules.
7. Database Language SQL. Standard ISO/IEC 9075-1, -2, etc. Available at: <http://www.iso.ch>.
8. Database Language SQL. Standard ANSI/ISO/IEC 9075-1, -2, etc. Available at: <http://www.ansi.org>.
9. Feuerstein S. and Pribyl B. Oracle PL/SQL Programming, O'Reilly, Sebastopol, CA, 2005.
10. Melton J. Advanced SQL:1999—Understanding Object-Relational and Other Advanced Features. Morgan Kaufmann, San Francisco, CA, 2003.
11. Melton J. and Simon A.R. SQL:1999—Understanding Relational Language Components. Morgan Kaufmann, San Francisco, CA, 2002.
12. Microsoft SQL Server Development Center. Transact-SQL Reference. <http://msdn2.microsoft.com/en-us/library/ms189826.aspx>.
13. Sanders R.E. ODBC 3.5 Developer’s Guide. McGraw-Hill, NY, 1998.
14. Sun Developer Network. JDBC Overview. <http://java.sun.com/products/jdbc/overview.html>.

SQL Isolation Levels

PHILIP A. BERNSTEIN

Microsoft Corporation, Redmond, WA, USA

Synonyms

Degrees of consistency; Degrees of isolation

Definition

A transaction is an execution of a well-defined set of read and write operations on shared data, which terminates with a commit operation that makes its updates permanent, or an abort operation that undoes its updates. Isolation levels define the situations in which a transaction can be affected by the execution of other transactions. In the ACID properties, isolation requires that transactions behave serializably, that is, as if they executed in a serial order with no interleaving. To obtain a serializable execution when many transactions are executing concurrently, a transaction's operations may be delayed and occasionally even rejected. This reduces the rate at which transactions execute. Users often regard this throughput reduction as unsatisfactory and therefore seek isolation levels that are less stringent than serializability, some of which are defined as part of the SQL language. These are the SQL Isolation Levels, which are called Read Uncommitted, Read Committed, Repeatable Read, and Serializable. They are derived primarily from the "degrees of consistency" originally presented in [3].

Key Points

The SQL isolation levels are defined in terms of the following types of interleavings, P1 – P3, that each isolation allows or prohibits. The interleavings are expressed by sequences of operations, using the notation $r_1[x]$ (respectively, $w_1[x]$) to represent the execution of a read (respectively write) operation by transaction T_1 on row x .

P1. Dirty Read – A dirty read is an operation that reads a value that was written by an uncommitted transaction. In the execution " $w_1[x] r_2[x]$," $r_2[x]$ performs a dirty read. The problem is that transaction T_1 might abort after T_2 read row x , in which case T_2 has read a value of x that never existed.

P2. Non-repeatable Read – A transaction reads a row x , a second transaction writes into x and commits,

and then the first transaction reads x again. In the execution " $w_0[x] r_1[x] w_2[x] c_2 r_1[x]$," transaction T_1 has experienced a non-repeatable read, since it read one value of x before T_2 executed and a different value of x after T_2 committed.

P3. Phantom – A transaction T_1 reads a set of rows that satisfy a predicate P (such as a WHERE-clause in SQL). Before T_1 commits or aborts, a second transaction T_2 inserts, updates, or deletes rows that change the set of rows that satisfy P . Therefore, if T_1 re-executes its read, the read will return a different set of rows than it returned the first time. The rows that appear and disappear are called phantoms. This is the same situation as non-repeatable reads, except that the set of rows that T_1 retrieves is affected by T_2 , not just their value. In the following execution, the row inserted by w_2 is a phantom:

- $r_1[\text{rows of the Employee table where Department = "Toy"}]$
- $w_2[\text{insert a new row in the Employee table where Department = "Toy"}]$
- $r_1[\text{rows of the Employee table where Department = "Toy"}]$

The SQL isolation levels are defined using the above three phenomena:

1. Read Uncommitted – All three phenomena (P1, P2, and P3) are allowed. The intent is to allow all interleavings of reads and writes by different transactions.
2. Read Committed – Dirty reads are prohibited. This ensures that each read operation reads a value that will not be undone because the transaction that wrote the value later aborts.
3. Repeatable Read – Dirty reads and non-repeatable reads are prohibited. The intent is to ensure that transaction executions are serializable except for phantom situations that arise.
4. Serializable – All three phenomena are prohibited. The intent is to ensure that transactions are truly serializable.

Each transaction may independently define its isolation level. This creates some difficulty in how a user should interpret the levels. For example, if a transaction runs as Serializable, it may still read data that was written by transactions running (say) as Read Committed.

SQL isolation levels have been criticized because there is a gap between the definition of the phenomena they prevent, and the intent of the isolation level as presented in the descriptions of (1) – (4) above. For details, see [1].

In principle, all transactions that perform updates should execute at the Serializable level. That way, if each transaction preserves database consistency, then each serial execution will preserve consistency. Hence a serializable execution will too. If update transactions execute at less than serializable level, then this consistency preservation guarantee is lost. Nevertheless, it is believed that most database applications execute at lower isolation levels than Serializable, typically Read Committed. This gives them higher throughput at the expense of some loss of correctness. One simulation study of transaction performance showed that transaction throughput is 2½–3 times higher with transactions executing at Read Committed level compared to Serializable level [2].

Cross-references

- ▶ [ACID Properties](#)
- ▶ [Serializability](#)
- ▶ [Transaction](#)
- ▶ [Transaction Model](#)

Recommended Reading

1. Berenson H., Bernstein P., Gray J., Melton J., O’Neil E., and O’Neil P. A critique of ANSI SQL isolation levels. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 1–10.
2. Bober P.M. and Carey M.J. On mixing queries and transactions via multiversion locking. In Proc. 8th Int. Conf. on Data Engineering, 1992, pp. 548–556.
3. Gray J., Lorie R.A., Potzculo G.R., and Traiger I.L. Granularity of locks and degrees of consistency in a shared database. In IFIP Working Conf. on Modelling in Data Base Management Systems, 1976, pp. 365–394. Reprinted in Readings in Database Systems (3rd edn.), M. Stonebraker and J. Hellerstein (eds.). Morgan Kaufmann, 1998, pp. 175–193.
4. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993, pp. 397–403.

SQL-Based Temporal Query Languages

MICHAEL BÖHLEN¹, JOHANN GAMPER¹,
CHRISTIAN S. JENSEN², RICHARD T. SNODGRASS³
¹Free University of Bozen-Bolzano, Bolzano, Italy
²Aalborg University, Aalborg, Denmark
³University of Arizona, Tucson, AZ, USA

Definition

More than two dozen extensions to the relational data model have been proposed that support the storage

and retrieval of time-referenced data. These models timestamp tuples or attribute values, and the timestamps used include time points, time periods, and finite unions of time periods, termed temporal elements.

A temporal query language is defined in the context of a specific data model. Most notably, it supports the specification of queries on the specific form of time-referenced data provided by its data model. More generally, it enables the management of time-referenced data.

Different approaches to the design of a temporal extension to the Structured Query Language (SQL) have emerged that yield temporal query languages with quite different design properties.

Historical Background

A number of past events and activities that included the temporal database community at large had a significant impact on the evolution of temporal query languages. The 1987 *IFIP TC 8/WG 8.1 Working Conference on Temporal Aspects in Information Systems* [7] covered topics such as requirements for temporal data models and information systems, temporal query languages, versioning, implementation techniques, as well as temporal logic, constraints, and relations to natural language.

The 1993 *ARPA/NSF International Workshop on an Infrastructure for Temporal Databases* [8] gathered researchers in temporal databases with the goal of consolidating the different approaches to temporal data models and query languages. In 1993, the influential collection *Temporal Databases: Theory, Design, and Implementation* [11] was also published. This collection describes a number of data models and query languages produced during the previous 10 years of temporal database research.

Year 1995 saw the publication of the book *The TSQL2 Temporal Query Language* [9]. TSQL2 represents an effort to design a consensus data model and query language, and it includes many of the concepts that were proposed by earlier temporal data models and query languages. In 1995, the *International Workshop on Temporal Databases* [3] was co-located with the VLDB conference.

Then, in 1996, *SQL/Temporal: Part 7 of SQL3* was accepted. This was the result of an effort aimed at transferring results of temporal database research into SQL3. The first step was a proposal of a new part to SQL3, termed SQL/Temporal, which included the PERIOD data type. In 1997, the *Dagstuhl Seminar*

on *Temporal Databases* took place [5]. Its goal was to discuss future directions for temporal database management, with respect to both research issues and the means to incorporate temporal databases into mainstream application development.

Foundations

A discrete and totally ordered time domain is assumed that consists of time instants/points. The term (time) “period” is used to denote a convex subset of the time domain. The term (time) “interval” then denotes a duration of time, which coincides with its definition in SQL. As a running example, the temporal relation *Rental* in Fig. 1(a) is used, which records car rentals, e.g., customer C101 rents vehicle V1234 from time 3 to time 5. Figures 1(b)–(e) show different representations of this relation, using the strong period-based, weak period-based, point-based, and parametric model, respectively. (In all the example relation instances, the conventional attribute(s) are separated from the timestamp attribute(s) with a vertical line.) The following queries together with their intended results build on the car rental example. These will serve for illustration.

Q1: *All rentals that overlap the time period [7,9].* Query Q1 asks for all available information about rentals that overlap the period [7,9].

CustID	VID	T
C102	V1245	[5,7]
C102	V1234	[9,12]

Q2: *All 2-day rentals.* This query constrains the number of time points included in a time period and teases out the difference between the use of time points versus periods.

VID	T
V1245	[19,20]
V1245	[21,22]

Q3: *How many vehicles have been rented?* This is an example of an ordinary query that must be applied to each state of a temporal database. The non-temporal query is an aggregation. Thus, the result at a specific time point is computed over all tuples that are valid

at that time point. (Note that some query languages don't return tuples when there is no data, e.g., when the Cnt is 0.)

Cnt	T
1	[3,4]
2	[5,5]
1	[6,7]
0	[8,8]
1	[9,12]
0	[13,18]
1	[19,20]
1	[21,22]

Q4: *How many rentals were made in total?* This is another aggregation query; however, the aggregation is to be applied independently of any temporal information.

Cnt
5

Q5: *List all (current) rentals.* This query refers to the (constantly moving) current time. It is assumed that the current time is 5.

CustID	VID
C101	V1234
C102	V1245

Approach I: Abstract Data Types – SQL/ATD

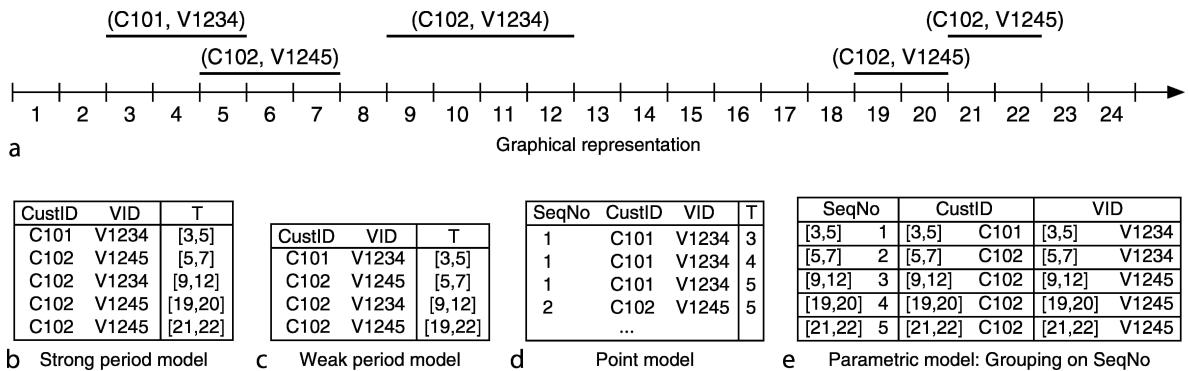
The earliest and, from a language design perspective, simplest approaches to improving the temporal data management capabilities of SQL have simply introduced time data types and associated predicates and functions. This approach is illustrated on the *Rental* instance in Figure 1(b).

Q1_{SQL/ATD}: select * from Rental where T overlaps [7,9]

Q2_{SQL/ATD}: select VID, T from Rental where duration(T) = 2

Q4_{SQL/ATD}: select count(*) as Cnt from Rental

Q5_{SQL/ATD}: select CustID, VID from Rental where T overlaps [now, now]



SQL-Based Temporal Query Languages. Figure 1. Temporal relation *rental*.

The predicates on time-period data types available in query languages have been influenced by Allen’s 13 period relationships [1], and different practical proposals for collections of predicates exist. For example, the overlaps predicate (as defined in the TSQL2 language) can be used to formulate Query Q1. Predicates that limit the duration of a period (Q2) and retrieve current data (Q5) follow the same approach.

Expressing the time-varying aggregation of Q3 in SQL is possible, but exceedingly complicated and inefficient. The hard part is that of expressing the computation of the periods during which the aggregate values remain constant. (This requires about two dozen lines of SQL with nested NOT EXISTS subqueries [10, pp. 165–166].) In contrast, counting the rentals independently of the time references is easy, as shown in Q4.

Adding a new ADT to SQL has limited impact on the language design, and extending SQL with new data types with accompanying predicates and functions is relatively simple and fairly well understood. The approach falls short in offering means of conveniently formulating a wide range of queries on period timestamped data, including temporal aggregation. It also offers no systematic way of generalizing a simple snapshot query to becoming time-varying. Shortcomings such as these motivate the consideration of other approaches.

Approach II: Folding and Unfolding – IXSQL

Another approach is to equip SQL with the ability to normalize timestamps. Advanced most prominently by Lorentzos [4,6] in the IXSQL language, the earliest and most radical approach is to introduce two

functions: `unfold`, which decomposes a period-timestamped tuple into a set of point-timestamped tuples, and `fold`, which “collapses” a set of point-timestamped tuples into value-equivalent tuples timestamped with maximum periods. The general pattern for queries is then: (i) construct the point-based representation by unfolding the argument relation(s), (ii) compute the query on the period-free representation, and (iii) fold the result to obtain a period-based representation. The *Rental* relation in Fig. 1(b) is assumed.

```
Q3IXSQL: select count(*) as Cnt, T
  from (select * from Rental
         reformat as unfold T)
        group by T reformat as fold T
```

The IXSQL formulations of Q1, Q2, Q4, and Q5 are essentially those of the ADT approach (modulo minor syntactic differences); specifically, normalization is not needed. The `fold` and `unfold` functions become useful for the temporal aggregation in Q3. The inner query unfolds the argument relation, yielding the point-based representation in Fig. 1(d), on which the aggregation is computed. The `fold` function then transforms the result back into a period-stamped relation, which, however, is different from the intended result because the last two tuples are merged into a single tuple (1,[19,22]). The combination of unfolding and folding yields maximal periods of snapshot equivalent tuples and does not carry over any lineage information.

SQL with folding and unfolding is conceptually simple and offers a systematic approach to formulating at least some temporal queries, including temporal

queries that generalize non-temporal queries. It obtains the representational benefits of periods while avoiding the potential problems they pose in query formulation, since the temporal data are manipulated in point-stamped form. The `fold` and `unfold` functions preserve the information content in a relation only up to that captured by the point-based perspective; thus, lineage information is lost. This leaves some “technicalities” (which are tricky at times) to be addressed by the application programmer.

Approach III: Point Timestamps – SQL/TP

A more radical approach to designing a temporal query language is to simply assume that temporal relations use point timestamps. The temporal query language SQL/TP advanced by Toman [12] takes this approach to generalizing queries on non-temporal relations to apply to temporal relations. The point-timestamped `Rental` relation in Fig. 1(d) is assumed in the following.

```
Q1SQL/TP: select distinct a.* from Rental a,
            Rental b where a.SeqNo = b.SeqNo
            and or (b.T = 7 or b.T = 8 or b.T = 9)
Q2SQL/TP: select SeqNo, VID, T from Rental
            group by SeqNo having count(T) = 2
Q3SQL/TP: select count(*) as Cnt, T
            from Rental group by T
Q4SQL/TP: select count(distinct SeqNo) as Cnt
            from Rental
Q5SQL/TP: select CustID, VID from Rental
            where T = now
```

Q1 calls for a comparison of neighboring database states. The point-based perspective, which separates the database states, does not easily support such queries, and a join is needed to report the original rental periods. The `distinct` keyword removes duplicates that are introduced if a tuple shares more than one time point with the period [7,9].

Duration queries, such as Q2, are formulated as aggregations and require an attribute, in this case `SeqNo`, that distinguishes the individual rentals. The strength of SQL/TP is in its generalization of queries on snapshot relations to queries on temporal relations, as exemplified by Q3. The general principle is to extend the snapshot query to separate database snapshots, which here is done by the grouping clause. SQL/TP and SQL are opposites when it comes to the handling of temporal information. In SQL, *time-varying*

aggregation is poorly supported, while SQL/TP needs an additional attribute that identifies the real-world facts in the argument relation to support *time-invariant* aggregation (Q4).

The restriction to time points ensures a simple and well-defined semantics that avoids many of the pitfalls that can be attributed to period timestamps. As periods are still to be used in the physical representation and user interaction, one may think of SQL/TP as a variant of IXSQL where, conceptually, queries must always apply `unfold` as the first operation and `fold` as the last. To express the desired queries, an identifying attribute (e.g., `SeqNo`) is often needed. Such identifiers do not offer a systematic way of obtaining point-based semantics *and* a semantics that preserves the periods of the argument relations. The query “When was vehicle V1245, but not vehicle V1234, rented?” illustrates this point. A formulation using the temporal difference between the timestamp attributes does not give the expected answer {[6,7],[19,20], [21,22]} because the sequence number is not included. If the sequence number is included, the difference is effectively disabled. This issue is not only germane to SQL/TP, but applies equally to all approaches that use a point-based data model.

Approach IV: Syntactic Defaults – TSQL2

What may be viewed as syntactic defaults have been introduced to make the formulation of common temporal queries more convenient. The most comprehensive approach based on syntactic defaults is TSQL2 [9]. As TSQL2 adopts a point-based perspective, the `Rental` instance in Fig. 1(c) is assumed, where the periods are a shorthand representation of time points.

```
Q1TSQL2: select * from Rental
            where valid(Rental) overlaps
            period '7-9'
Q2TSQL2: select SeqNo, VID from Rental
            where cast(valid(Rental) as
            interval) = 2
Q3TSQL2: select count(*) as Cnt from Rental
            group by valid(Rental) using instant
Q4TSQL2: select snapshot count(*) as Cnt
            from Rental
Q5TSQL2: select snapshot * valid(date 'now')
            from Rental
```

In TSQL2, a `valid` clause, which by default is present implicitly after the `select` clause, computes the

intersection of the valid times of the relations in the `from` clause, which is then returned in the result. With only one relation in the `from` clause, this default clause yields the original timestamps as exemplified in Q1 and Q2. The cast function in Q2 maps between periods (e.g., [7–9]) and intervals (e.g., 3 days). The argument relation must be augmented by the `SeqNo` attribute (thus obtaining a relation with five tuples, as in Fig. 1(b)) for this query to properly return the 2-day rentals.

The default behavior of the implicit `valid` clause was designed with snapshot reducibility in mind, which shows nicely in the instant temporal aggregation query Q3. The grouping is performed according to the time points, not the original timestamps returned by `valid(Rental)`. The using instant is in fact the default and could be omitted (added for clarity). As TSQL2 returns temporal relations by default, the `snapshot` keyword is used in queries Q4 and Q5 to retrieve non-temporal relations.

Well-chosen syntactic defaults yield a language that enables succinct formulation of common temporal queries. However, adding temporal support to SQL in this manner is difficult since the non-temporal constructs do not permit a systematic and easy way to express the defaults. It is challenging to be comprehensive in the specification of such defaults, and to ensure that they do not interact in unattractive ways. Thus, syntactic defaults lack “scalability” over language constructs.

Approach V: Statement Modifiers – ATSQL

ATSQL [2] introduces temporal statement modifiers to offer a systematic means of constructing temporal queries from non-temporal queries. A temporal query is formulated by first formulating the corresponding non-temporal query, and then prepending this query with a statement modifier that tells the database system to use temporal semantics. In contrast to syntactic defaults, statement modifiers are semantic in that they apply in the same manner to any statement they modify. The strong period-timestamped `Rental` instance in Fig. 1(b) is assumed in the following.

```
Q1ATSQL: seq vt select * from Rental
           where T overlaps [7,9]
Q2ATSQL: seq vt select VID from Rental
           where duration(T) = 2
Q3ATSQL: seq vt select count(*) as Cnt from
           Rental
```

Q4^{ATSQL}: nseq vt select count(*) as Cnt

from Rental

Q5^{ATSQL}: select * from Rental

Queries Q1 and Q2 can be formulated almost as in SQL. The `seq vt` (“sequenced valid time”) modifier indicates that the semantics is consistent with evaluating the non-temporal query on a sequence of non-temporal relations, and ensures that the original timestamps are returned. Modifiers also work for queries that use period predicates, such as, e.g., Allen’s relations, which cannot be used in languages of point-timestamped data models.

Query Q3 is a temporal generalization of a non-temporal query and can be formulated by prepending the non-temporal SQL query with the `seq vt` modifier. The modifier ensures that at each time point, the aggregates are evaluated over all tuples that overlap with that time point. Query Q4 is to be evaluated independently of the time attribute values of the tuples. This is achieved by using the `nseq vt` (“non-sequenced valid time”) modifier, which indicates that what follows should be treated as a regular SQL query.

A query without any modifiers considers only the current states of the argument relations, as exemplified by Query Q5. This ensures that legacy queries on non-temporal relations are unaffected if the non-temporal relations are made temporal.

Statement modifiers are orthogonal to SQL and adding them to SQL represents a much more fundamental change to the language than, e.g., adding a new ADT or syntactic defaults. The notion of statement modifiers offers a wholesale approach to rendering a query language temporal: modifiers control the semantics of any query language statement. This language mechanism is independent of the syntactic complexity of the queries that the modifiers are applied to. It becomes easy to construct temporal queries that generalize snapshot queries.

Approach VI: Temporal Expressions – TempSQL

The notion of temporal expression was originally advocated by Gadia and is supported in the TempSQL language [11, p. 28ff], which is based on the parametric data model (see Fig. 1(e)). Relations in TempSQL consist of tuples with attribute values that are functions from a subset of the time domain to some value domain (specified as a pair of a temporal

element, a finite union of time periods, and a value). The functions in the same tuple must have the same domain. The relations are keyed. If a set of attributes is a key, then no two tuples are allowed to exist in the relation that have the same range values for those attributes. Fig. 1(e) with the key SeqNo is assumed in the following.

```

Q1TempSQL: select * from Rental
    where [[VID]] ∩ [7,9] ≠ ∅
Q2TempSQL: select VID from Rental
    where duration([[VID]]) = 2
Q3TempSQL: select count(*) as Cnt from Rental
Q5TempSQL: select * from Rental

```

Queries Q1 and Q2 can be formulated using temporal expressions. If X is an expression that returns a function from time to some value domain then $[[X]]$ is a temporal expression which returns the domain of X , i.e., the time when X is true. The result of Q2 is the relation $\{([19,22] \text{ V1245})\}$. For the aggregation query Q3, TempSQL automatically performs an instant temporal aggregation [11, p. 42]. A different query must be used to determine the time-invariant count in Q4. One possibility would be to formulate a query that first drops or equalizes all timestamps and then performs the above aggregation. For so-called current users, TempSQL offers built-in support for accessing the current state of a database, by assuming that the argument relations are the ordinary snapshot relations that contain the current states of the temporal relations. This is exemplified in Q5.

Temporal expressions as used in TempSQL, which return the temporal elements during which a logical expression is true, are convenient and often enable the elegant formulation of queries. Temporal expressions along with temporal elements fit well into the point-based framework. However, as of yet, little research has been done to further explore temporal expressions and to include them into query languages.

Key Applications

SQL-based temporal query languages are intended for use in database applications that involve the management of time-referenced data. Such applications are found literally in all data management application areas – in fact, virtually all real-world databases contain time-referenced data. SQL-based languages are attractive in comparison to other types of languages

because SQL is used by existing database management systems.

Future Directions

While temporal query language support appears to be emerging in commercial systems, comprehensive temporal support is still not available in products.

Much research in temporal query languages has implicitly or explicitly assumed a traditional administrative data management setting, as exemplified by the car rental example. The design of temporal query languages for other kinds of data and applications, e.g., continuous sensor data, has received little attention.

Cross-references

- ▶ [Allen's Relations](#)
- ▶ [Now in Temporal Databases](#)
- ▶ [Period-Stamped Temporal Models](#)
- ▶ [Point-Stamped Temporal Models](#)
- ▶ [Temporal Database](#)
- ▶ [Temporal Data Models](#)
- ▶ [Temporal Element](#)
- ▶ [Time Interval](#)
- ▶ [Time Period](#)
- ▶ [Temporal Query Languages](#)
- ▶ [TSQL2](#)
- ▶ [Valid Time](#)

Recommended Reading

1. Allen J.F. Maintaining Knowledge about temporal intervals. *Commun. ACM*, 26 (11):832–843, 1983.
2. Böhlen M.H., Jensen C.S., and Snodgrass R.T. Temporal statement modifiers. *ACM Trans. on Database Syst.*, 25(4):407–456, 2000.
3. Clifford J. and Tuzhilin A. editors. Recent advances in temporal databases. In Proc. Int. Workshop on Temporal Databases, 1995.
4. Date C.J., Darwen H., and Lorentzos N. editors. *Temporal Data and the Relational Model*. Morgan Kaufmann publishers, 2002.
5. Etzion O., Jajodia S., and Sripada S. editors. *Temporal Databases: Research and Practice*, Volume 1399 of Lecture Notes in Computer Science. Springer Verlag, 1998.
6. Lorentzos N.A. and Johnson R.G. Extending relational algebra to manipulate temporal data. *Inf. Syst.*, 13(3):289–296, 1988.
7. Rolland C., Bodart F., and Léonard M. editors. Temporal aspects in information systems. In Proc. IFIP TC 8/WG 8.1 Working Conf. on Temporal Aspects in Information Systems, 1987.
8. Snodgrass R.T. editor. In Proc. Int. Workshop on an Infrastructure for Temporal Databases, 1993.
9. Snodgrass R.T. editor. *The TSQL2 Temporal Query Language*. Kluwer, 1995.

10. Snodgrass R.T. Developing Time-Oriented Database Applications in SQL. Morgan Kaufmann Publishers, San Francisco, CA, July 1999.
11. Tansel A., Clifford J., Gadia S., Jajodia S., Segev A., and Snodgrass R.T. Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings Publishing Company, Inc. 1993.
12. Toman D. Point-based temporal extensions of SQL and their efficient implementation. In [5], 1997, pp. 211–237.

SRM

- Storage Resource Management

Stability-based Validation of Clustering

- Clustering Validity

Stable Distribution

PING LI

Cornell University, Ithaca, NY, USA

Synonyms

Lévy skew α -stable distribution

Definition

A random variable Z is said to follow a symmetric α -stable distribution[13,15], where $0 < \alpha \leq 2$, if the Fourier transform of its probability density function $f_Z(z)$ satisfies

$$\int_{-\infty}^{\infty} e^{\sqrt{-1}zt} f_Z(z) dt = e^{-d|t|^{\alpha}}, \quad 0 < \alpha \leq 2 \quad (1)$$

where $d > 0$ is the scale parameter. This is denoted by $Z \sim S(\alpha, d)$.

There is an equivalent definition. A random variable Z follows a symmetric α -stable distribution if, for any real numbers, C_1 and C_2 ,

$$C_1 Z_1 + C_2 Z_2 \stackrel{d}{=} (|C_1|^\alpha + |C_2|^\alpha)^{1/\alpha} Z, \quad (2)$$

where Z_1 and Z_2 are independent copies of Z , and the symbol “ $\stackrel{d}{=}$ ” denotes equality in distribution.

The probability density function $f_Z(z)$ can be obtained by taking inverse Fourier transform of 1. In

particular, $f_Z(z)$ can be expressed in closed-forms when $\alpha = 2$ (i.e., the normal distribution) and $\alpha = 1$ (i.e., the Cauchy distribution).

Historical Background

The early comprehensive development of the stable distribution theory is credited to the French mathematician P. Lévy and the Soviet mathematician A. Ya. Khinchin, in the 1920s and 1930s[13,15]. Stable distributions have been widely used for modeling real-world data which exhibit *heavy-tailed* behaviors, for example, data that do not have finite variance or even finite mean.

In databases, data mining, and theoretical computer science, stable distributions have become an important tool for developing randomized dimension reduction algorithms, in particular, for efficiently computing summary statistics including distances, frequency moments, inner products and angles, in massive dynamic data sets. This type of technique is called (*stable*) *random projections*[1–4,7–9,11,14].

Pioneered by Alon, Matias, and Szegedy [2] in 1996, the method of *random projections* has been extensively applied in databases for approximating joint sizes and (l_2) Euclidean distances. Indyk and Motwani [8] in 1998 developed *local sensitive hashing (LSH)* using random projections, for efficiently searching for approximate nearest neighbors in high-dimensional data. In 2004, Datar et al. [4] extended LSH for approximating nearest neighbors in the l_α ($0 < \alpha \leq 2$) norm. Indyk[7] in 2000 proposed using stable random projections to approximate the l_α distances and frequency moments in massive data streams. Cormode et al. [3] in 2002 proposed approximating the (l_0) Hamming distances in dynamic data using stable random projections with very small α .

The method of stable random projections eventually boils down to estimating the scale parameter of an α -stable distribution for a fixed α . This problem has been studied in statistics. For example, Fama and Roll[6] suggested estimators based on sample quantiles. Ping Li [11] in 2008 developed estimators based on the geometric mean and the harmonic mean.

Foundations

From the definitions (1) and (2), it follows that, if D random variables, r_1, r_2, \dots, r_D , are independent and identically distributed (i.i.d.), $r_i \sim S(\alpha, 1)$, then a

linear combination of r_i 's also follows an α -stable distribution. That is

$$c_1 r_1 + c_2 r_2 + \dots + c_D r_D \sim S(\alpha, |c_1|^\alpha + |c_2|^\alpha + \dots + |c_D|^\alpha), \quad (3)$$

for any real numbers c_1, c_2, \dots, c_D . This property is the foundation for stable random projections.

Stable Random Projections

The basic procedure of stable random projection is quite straightforward. Consider two D -dimensional vectors, $u_1 \in \mathbb{R}^D$ and $u_2 \in \mathbb{R}^D$. A matrix $\mathbf{R} = \{r_{ij}\}_{i=1}^D \{j=1}^k \in \mathbb{R}^{D \times k}$ is generated by sampling the entries from i.i.d. α -stable distributions, i.e., $r_{ij} \sim S(\alpha, 1)$. The matrix-vector multiplications, $v_1 = \mathbf{R}^T \times u_1$ and $v_2 = \mathbf{R}^T \times u_2$, result in two k -dimensional vectors, $v_1 \in \mathbb{R}^k$ and $v_2 \in \mathbb{R}^k$. The entries of v_1 and v_2 also follow α -stable distributions:

$$v_{1,j} \sim S\left(\alpha, \sum_{i=1}^D |u_{1,i}|^\alpha\right), \text{ i.i.d. } j = 1, 2, \dots, k \quad (4)$$

$$v_{2,j} \sim S\left(\alpha, \sum_{i=1}^D |u_{2,i}|^\alpha\right), \text{ i.i.d. } j = 1, 2, \dots, k \quad (5)$$

$$v_{1,j} - v_{2,j} \sim S\left(\alpha, \sum_{i=1}^D |u_{1,i} - u_{2,i}|^\alpha\right), \text{ i.i.d. } j = 1, 2, \dots, k \quad (6)$$

The term, $\sum_{i=1}^D |u_{1,i}|^\alpha$, is the l_α norm (raised to the α th power) of the vector u_1 ; and in data stream computations, it is often referred to as the α th frequency moment. The term, $\sum_{i=1}^D |u_{1,i} - u_{2,i}|^\alpha$, is the l_α distance (raised to the α th power) between vectors u_1 and u_2 . Many applications such as clustering, classification, nearest neighbor searching etc. only require pairwise distances of the data; and hence one might discard the original massive data after stable random projections, as long as one can estimate the distances from the stable samples.

Statistical Estimations

The method of stable random projections boils down to a statistical estimation problem. That is, given k i.i.d. samples $x_j \sim S(\alpha, d_{(\alpha)})$, $j = 1, 2, \dots, k$, estimate the scale parameter $d_{(\alpha)}$. Listed below are various estimators, together with their estimation variances either exactly or asymptotically.

- The arithmetic mean estimator, for $\alpha = 2$ only

$$\hat{d}_{(2),am} = \frac{1}{k} \sum_{j=1}^k |x_j|^2, \quad (7)$$

$$\text{Var}\left(\hat{d}_{(2),am}\right) = \frac{2}{k} d_{(2)}^2. \quad (8)$$

- The harmonic mean estimator, for small α only

$$\begin{aligned} \hat{d}_{(\alpha),hm} &= \frac{-\frac{2}{\pi} \Gamma(-\alpha) \sin\left(\frac{\pi}{2}\alpha\right)}{\sum_{j=1}^k |x_j|^{-\alpha}} \\ &\left(k - \left(\frac{-\pi \Gamma(-2\alpha) \sin(\pi\alpha)}{[\Gamma(-\alpha) \sin(\frac{\pi}{2}\alpha)]^2} - 1 \right) \right), \end{aligned} \quad (9)$$

$$\begin{aligned} \text{Var}\left(\hat{d}_{(\alpha),hm}\right) &= d_{(\alpha)}^2 \frac{1}{k} \left(\frac{-\pi \Gamma(-2\alpha) \sin(\pi\alpha)}{[\Gamma(-\alpha) \sin(\frac{\pi}{2}\alpha)]^2} - 1 \right) \\ &+ O\left(\frac{1}{k^2}\right). \end{aligned} \quad (10)$$

As α approaches zero, in the limit,

$$\begin{aligned} \lim_{\alpha \rightarrow 0+} -\frac{2}{\pi} \Gamma(-\alpha) \sin\left(\frac{\pi}{2}\alpha\right) &= 1, \\ \lim_{\alpha \rightarrow 0+} \left(\frac{-\pi \Gamma(-2\alpha) \sin(\pi\alpha)}{[\Gamma(-\alpha) \sin(\frac{\pi}{2}\alpha)]^2} - 1 \right) &= 1. \end{aligned} \quad (11)$$

- The geometric mean estimator

$$\hat{d}_{(\alpha),gm} = \frac{\prod_{j=1}^k |x_j|^{\alpha/k}}{\left[\frac{2}{\pi} \Gamma\left(\frac{2}{k}\right) \Gamma\left(1 - \frac{1}{k}\right) \sin\left(\frac{\pi\alpha}{2k}\right)\right]^k}. \quad (12)$$

$$\text{Var}\left(\hat{d}_{(\alpha),gm}\right) = d_{(\alpha)}^2 \left\{ \frac{\left[\frac{2}{\pi} \Gamma\left(\frac{2\alpha}{k}\right) \Gamma\left(1 - \frac{2}{k}\right) \sin\left(\frac{\pi\alpha}{k}\right)\right]^k}{\left[\frac{2}{\pi} \Gamma\left(\frac{2}{k}\right) \Gamma\left(1 - \frac{1}{k}\right) \sin\left(\frac{\pi\alpha}{2k}\right)\right]^{2k}} - 1 \right\} \quad (13)$$

$$= d_{(\alpha)}^2 \frac{1}{k} \frac{\pi^2}{12} (\alpha^2 + 2) + O\left(\frac{1}{k^2}\right). \quad (14)$$

- The sample median estimator

$$\hat{d}_{(\alpha),me} = \frac{\text{median}\{|x_j|^\alpha, j = 1, 2, \dots, k\}}{\text{median}\{S(\alpha, 1)\}^\alpha}. \quad (15)$$

The estimation variance of the sample median estimator $\hat{d}_{(\alpha),me}$ cannot be expressed in closed-forms.

Compared with the geometric mean estimator, the sample median estimator is not as accurate when the sample size k is not very large. The sample median estimator, however, is more convenient to compute.

Sample Complexity

When $\alpha = 2$, the celebrated Johnson-Lindenstrauss (JL) Lemma [9] showed that k , the required number of projections, should satisfy $k = O(\log n/\epsilon^2)$ so that any pairwise l_2 distance among n data points can be approximated within a $1 \pm \epsilon$ factor of the truth.

For general $0 < \alpha \leq 2$, it is proved[11] using the geometric mean estimator that the sample complexity should also be $k = O(\log n/\epsilon^2)$. The constants can be explicitly specified.

Sampling from Stable Distributions

Sampling from a stable distribution is in general quite expensive, unless $\alpha = 2$ or $\alpha = 1$. One procedure is described in [13, Proposition 1.71.1]. A random variable W_1 is sampled from a uniform distribution on the interval $(-\frac{\pi}{2}, \frac{\pi}{2})$; and a random variable E_1 is sampled from an exponential distribution with mean 1. W_1 and E_1 are independent. Then

$$\frac{\sin(\alpha W_1)}{\cos(W_1)^{1/\alpha}} \left(\frac{\cos((1-\alpha)W_1)}{E_1} \right)^{(1-\alpha)/\alpha} \quad (16)$$

is distributed as $S(\alpha, 1)$.

Under certain reasonable regularity assumptions on the original data, it is possible to simplify the sampling procedure by replacing the α -stable distribution $S(\alpha, 1)$ with a mixture of a symmetric α -Pareto distribution (with probability $0 < \beta \leq 1$) and a point mass at the origin (with probability $1 - \beta$), i.e.,

$$\begin{cases} P_\alpha & \text{with prob. } \frac{\beta}{2} \\ 0 & \text{with prob. } 1 - \beta, \\ -P_\alpha & \text{with prob. } \frac{\beta}{2} \end{cases} \quad (17)$$

where P_α denotes an α -Pareto variable, i.e., $\Pr(P_\alpha > t) = \frac{1}{t^\alpha}$ if $t \geq 1$, and 0 otherwise. An α -Pareto distribution has the same tail behaviors as $S(\alpha, 1)$, but it is much easier to sample from. For example, given a random variable U drawn from a uniform distribution on the unit interval $(0,1)$, then $1/U^{1/\alpha}$ follows an α -Pareto distribution.

If D random variables, r_1, r_2, \dots, r_D , are sampled i.i.d. from (17), then a linear combination of r_i 's is asymptotically stable, as $D \rightarrow \infty$. That is [10],

$$c_1 r_1 + c_2 r_2 + \dots + c_D r_D \Rightarrow$$

$$S\left(\alpha, \beta \Gamma(1-\alpha) \cos\left(\frac{\pi}{2}\alpha\right) \sum_{j=1}^D |c_j|^\alpha\right), \quad (18)$$

provided that the data, c_1, c_2, \dots, c_D , satisfy

$$\frac{\max_{1 \leq i \leq D} |c_i|}{\left(\sum_{i=1}^D |c_i|^\alpha\right)^{1/\alpha}} \rightarrow 0, \text{ as } D \rightarrow \infty. \quad (19)$$

The parameter β in (17) controls the sparsity of the projection matrix. Small β values considerably reduce the processing cost for conducting random projections. β should be chosen according to the data dimension D and the prior knowledge about the data. Some synthetic and real-world data experiments in [10] indicated that, when $\alpha = 1$, using (17) with $\beta < 0.1$, achieved very similar estimation accuracy as using the exact stable distribution, even when D is not too large.

Key Applications

Stable distributions have been widely used for modeling real-world *heavy-tailed* data, arising in finance, economics, Internet traffic, computational Linguistics, and many other fields.

There have been numerous applications of stable random projections, in theoretical computer science, databases, data mining, data streams, and signal recovery[5].

Stable Random Projections for Dimension Reductions

Data mining and machine learning algorithms often assume a “data matrix” $\mathbf{A} \in \mathbb{R}^{n \times D}$, with n rows and D columns. For many algorithms, the data matrix \mathbf{A} is utilized only through pairwise distances of \mathbf{A} instead of the original data. A projection matrix $\mathbf{R} \in \mathbb{R}^{D \times k}$ is generated by sampling each entry from i.i.d. $S(\alpha, 1)$. The projected data matrix $\mathbf{B} = \mathbf{A} \times \mathbf{R} \in \mathbb{R}^{n \times k}$ contain enough information to approximately recover pairwise l_α distances of \mathbf{A} . The number of projections (sample size), should satisfy $k = O(\log n/\epsilon^2)$.

- The original data matrix \mathbf{A} may be too large for physical memory, for example, \mathbf{A} could be the term-by-document matrix at Web scale. Even if \mathbf{A} may fit in memory, storing all pairwise distances of \mathbf{A} in memory can be infeasible when $n > 10^6$. In contrast, the projected data matrix \mathbf{B} may be small

enough for the memory. Because \mathbf{B} has only k columns, pairwise distances may be computed on demand.

- Computing all pairwise distances of \mathbf{A} costs $O(n^2D)$. The cost is reduced to $O(nDk + n^2k)$ using stable random projections.
- When $\alpha = 2$, the projected data matrix \mathbf{B} preserve not only the pairwise (squared) l_2 distances of \mathbf{A} in expectations, but also the pairwise inner products of \mathbf{A} in expectations. Some applications care about inner products more than distances. In databases, for example, counting the joint sizes can be viewed as computing inner products.

Stable Random Projections for Data Stream Computations

Consider the *Turnstile* model [12], which is a linear model for data streams. The input data stream $s_t = (i, I_t)$ arriving sequentially describes the underlying signal S , meaning $S_t[i] = S_{t-1}[i] + I_t$, $i = 1$ to D , where t denotes time. For example, S may represent the arriving IP addresses ($D = 2^{64}$) and $S_t[i]$ records the frequencies of IP address i . The term $\sum_{i=1}^D |S_t[i]|^\alpha$ is often referred to as the α th frequency moment of S_t . Due to the linearity of the *Turnstile* model, stable random projections can be applied for approximating the frequency moments.

Again, a random projection matrix $\mathbf{R} \in \mathbb{R}^{D \times k}$ is generated by sampling each entry r_{ij} from i.i.d. $S(\alpha, 1)$. A vector x of length k is initialized so that $x_j = 0$, for $j = 1$ to k . Then for each arriving tuple $s_t = (i, I_t)$, update $x_j \leftarrow x_j + r_{ij} \times I_t$ for $j = 1$ to k .

At any time t , the entries x_j , $j = 1$ to k , are i.i.d. samples from $S(\alpha, \sum_{i=1}^D |S_t[i]|^\alpha)$; and hence one can estimate the α th frequency moment $\sum_{i=1}^D |S_t[i]|^\alpha$. Due to the linearity, the same methodology can also be applied for approximating the difference between two streams.

In particular, when $\alpha \rightarrow 0+$, $\sum_{i=1}^D |S_t[i]|^\alpha$ approaches the Hamming norm of S_t , which is the total number of nonzero entries, sometimes referred to as the number of *distinct items*. Thus, stable random projections can provide the tool for approximating the Hamming norm (and Hamming distance) in dynamic streaming data, using very small α .

Cross-references

- ▶ Stream Mining

Recommended Reading

1. Achlioptas D. Database-friendly random projections: Johnson-Lindenstrauss with binary coins.. *J. Comput. Syst. Sci.*, 66(4): 671–687, 2003.
2. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th Annual ACM Symp. on Theory of Computing, 1996, pp. 20–29.
3. Cormode G., Datar M., Indyk P., and Muthukrishnan S. Comparing data streams using hamming norms (how to zero in). In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 335–345.
4. Datar M., Immorlica N., Indyk P., and Mirrokn V.S. Locality-sensitive hashing scheme based on p -stable distributions. In Proc. 20th Annual Symp. on Computational Geometry, 2004, pp. 253–262.
5. Donoho D.L. Compressed sensing. *IEEE Trans. Inform. Theory*, 52(4):1289–1306, 2006.
6. Fama E.F. and Roll R. Parameter estimates for symmetric stable distributions.. *J. Am. Stat. Assoc.*, 66(334):331–338, 1971.
7. Indyk P. Stable distributions, pseudorandom generators, embeddings, and data stream computation.. *J. ACM*, 53(3):307–323, 2006.
8. Indyk P. and Motwani R. Approximate nearest neighbors: Towards removing the curse of dimensionality. In Proc. 30th Annual ACM Symp. on Theory of Computing, 1998, pp. 604–613.
9. Johnson W.B. and Lindenstrauss J. Extensions of Lipschitz mapping into Hilbert space. *Contemp. Math.*, 26:189–206, 1984.
10. Li P. Very sparse stable random projections for dimension reduction in l_α ($0 < \alpha \leq 2$) norm. In Proc. 13th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2007.
11. Li P. Estimators and tail bounds for dimension reduction in l_α ($0 < \alpha \leq 2$) using stable random projections. In Proc. 19th Annual ACM -SIAM Symp. on Discrete Algorithms, 2008.
12. Muthukrishnan S. Data streams: Algorithms and applications. *Found. Trends Theor. Comput. Sci.*, 11:117–236, 2005.
13. Samorodnitsky G. and Taqqu M.S. *Stable Non-Gaussian Random Processes*. Chapman & Hall, 1994.
14. Vempala S. *The Random Projection Method*. American Mathematical Society, Providence, RI, 2004.
15. Zolotarev V.M. *One-dimensional Stable Distributions*. American Mathematical Society, Providence, RI, 1986.

Stack-based Query Language

KAZIMIERZ SUBIETA

Polish-Japanese Institute of Information Technology,
Warsaw, Poland

Synonyms

[SBQL](#)

Definition

Stack-Based Query Language [1] is a query and programming language devoted to object-oriented database

models. SBQL is the result of investigations into a uniform theoretical and conceptual basis for object-oriented query languages integrated with programming capabilities and abstractions, including database abstractions: updatable views, stored procedures and transactions. SBQL is developed according to the Stack-Based Architecture (SBA) [3,2] that is a conceptual frame for developing object-oriented query and programming languages. SBQL has the same role and meaning as object algebras, but it is much more universal and formally precise. SBQL deal with object store models that include complex objects, associations, classes, types, methods, inheritance, dynamic roles, encapsulation, polymorphism, semi-structured data, and other features. The functionality of SBQL includes all well-known query operators (selection, projection, navigation, path expressions, join, quantifiers, etc.), some less known operators (transitive closures, fixed-point equations, etc.), imperative (updating) statements integrated with queries, modules, procedures, functions and methods (with parameters being queries and recursive). SBQL deals with static strong type checking and with query optimization methods based on indices, rewriting rules and other techniques. Abstract implementation (a kind of operational semantics) is the basic paradigm of formal specification of SBQL semantics. It involves an abstract machine that employs abstract definitions of three internal data structures well-known in specification of programming languages: an *object store*, an *environmental stack*, and a *query result stack* (thus the Stack-Based Architecture). SBQL has been implemented within several research prototypes; the last and the most complete one is ODRA (Object Database for Rapid Applications).

Key Points

SBQL adopts a classical run-time mechanism of Programming Languages (PLs), with some improvements. The main syntactic decision is the unification of PL expressions and queries; queries remain the only kind of PL expressions. In SBQL there is no conceptual difference between expressions such as $2 + 2$ and $(x + y) \times z$, and queries such as *Employee* where *Salary* = 1000 or $(\text{Employee} \text{ where } \text{Salary} = (x + y) * z) . \text{Name}$. All such expressions/queries can be used as arguments of imperative statements, as parameters of procedures, functions or methods, as a return from a procedure, etc.

Semantically, SBQL is based on the classical *naming-scoping-binding* paradigm. Scopes are organized in an *environmental stack* with the “search from the top” rule. The operational semantics of query operators, programming constructs and procedures (functions, methods, views, etc.) is defined in terms of the three mentioned abstract data structures: *object store*, *environmental stack*, and *query results stack*. SBQL subdivides operators into algebraic and non-algebraic. Algebraic operators act only on the query result stack. The essence is non-algebraic operators, such as selection, projection, join, quantifiers, ordering, transitive closures, and iterations that are defined through the environment stack (with no reference to any object algebra or calculus). SBQL is extended by programming constructs, procedures, methods, modules, transactions and updatable views. SBQL has a (semi) strong static type checking and query optimizations. SBQL is implemented for different environments, including XML and workflow systems. SBQL is considered as a departure point for the new fourth generation database standard developed by OMG.

Cross-references

- ▶ [Class](#)
- ▶ [Database Programming Language](#)
- ▶ [Inheritance](#)
- ▶ [OODB \(Object-Oriented Database\)](#)
- ▶ [Query Language](#)
- ▶ [Query Optimization](#)
- ▶ [Strong Typing](#)
- ▶ [Updatable View](#)

Recommended Reading

1. SBQL web pages with a lot of resources and references. <http://www.sbql.pl/>.
2. Subieta K., Beeri C., Matthes F., Schmidt J.W. A stack-based approach to query languages. In Proc. 2nd East-West Database Workshop. 1994, pp. 159–180.
3. Subieta K., Kambayashi Y., Leszczyłowski J. Procedures in object-oriented query languages. In Proc. 21st Int. Conf. on Very Large Data Bases, 1995, pp. 182–193.

Staged Database Systems

- ▶ [Staged DBMS](#)

Staged DBMS

STAVROS HARIZOPOULOS
HP Labs, Palo Alto, CA, USA

Synonyms

Staged database systems

Definition

A Staged Database Management System (DBMS) is a database software architecture that optimizes data and instruction locality at all levels of the memory hierarchy in a computer system. An additional goal of Staged DBMS is to provide a robust and efficient platform for both parallelizing and pipelining database requests. The main principle of the Staged Database System design is to organize and assign software system components into self-contained stages; database request execution is broken into stages and sub-requests are group-processed at each stage. This allows for a context-aware execution sequence of requests that promotes reusability of both instructions and data, and also facilitates development of work sharing mechanisms, which has been a key application for StagedDB; work sharing is defined as any operation that reduces the total amount of work in a system by eliminating redundant computation or data accesses. Existing database systems can be converted to staged ones by carrying over their algorithms and mechanisms, and adapt those in a platform that supports staged execution.

Historical Background

Though the Staged DBMS architecture was proposed by Harizopoulos and Ailamaki in 2003 [3], one of the earliest prototype relational database systems, INGRES [12], also consisted of four “stages” (processes) that enabled pipelining; the reason for breaking up the DBMS software was main memory size limitations. Work in staged architectures re-emerged in the early 2000s, first by Larus and Parkes as a generic programming paradigm for building server applications [7], and subsequently by Welsh et al as a means for deploying highly concurrent internet services [13]. Initial prototypes of database systems developed at Carnegie Mellon University that followed the principles of StagedDB (Qpipe and Cordoba [4,5]), focused on the performance benefits of work sharing. A relational

engine based on the Staged DBMS design can proactively coordinate same-operator execution among concurrent queries, thereby exploiting common accesses to memory and disks as well as common intermediate result computation.

Foundations

Modern commercial DBMS are typically built as a large piece of software that serves multiple requests using a thread-based concurrency model. Queries are handled by one or more threads (or processes) that follow the query execution plan up to its completion. This model implicitly defines a query execution sequence and a resource utilization schedule in the system. Whenever a thread blocks due to an I/O, an ungranted lock request, an internal synchronization condition, or due to an expiring CPU time quantum, the thread scheduler assigns the CPU to the next runnable thread of the highest priority. This context-switching mechanism creates a logical gap in the sequence of actions the DBMS performs. While a software developer can optimize the individual steps involved in a single query’s execution, she or he typically has no means of applying similar optimization techniques to a collection of multiplexed queries.

A staged database system consists of a number of self-contained software modules, each encapsulated into a *stage*. A stage is an independent server with its own queue, thread support, and resource management that communicates and interacts with the other stages through a well-defined interface. Stages accept *packets*, each carrying a query’s state and private data, perform work on the packets, and may enqueue the same or newly created packets to other stages. Each stage is centered around exclusively owned (to the degree possible) server code and data. There are two levels of CPU scheduling: local thread scheduling within a stage and global scheduling across stages. The StagedDB design promotes stage autonomy, data and instruction locality, and minimizes the usage of global variables.

A stage provides two basic operations, enqueue and dequeue, and a queue for the incoming packets. The stage-specific server code is contained within dequeue. The system works through the exchange of packets between stages. A packet represents work that the server must perform for a specific query at a given stage. It first enters the stage’s queue through the

enqueue operation and waits until a dequeue operation removes it. Then, once the query's current state is restored, the stage specific code is executed. Depending on the stage and the query, new packets may be created and enqueued at other stages. Eventually, the stage code returns by either (i) destroying the packet (if done with that query at the specific stage), (ii) forwarding the packet to the next stage (i.e., from parse to optimize), or by (iii) enqueueing the packet back into the stage's queue (if there is more work but the client needs to wait on some condition). Queries use packets to carry their state and private data. Each stage is responsible for assigning memory resources to a query. In a shared-memory system, packets carry only pointers to the query's state and data structures (which are kept in a single copy). Each stage employs a pool of worker threads (the stage threads) that continuously call dequeue on the stage's queue, and one thread reserved for scheduling purposes (the scheduling thread). An analysis of scheduling tradeoffs in staged database systems along with a description of an initial implementation can be found in [2].

Key Applications

A key application for the Staged DBMS design has been detecting and exploiting work sharing opportunities at run-time inside a relational database engine. Traditional relational DBMS typically execute concurrent queries independently by invoking a set of operator instances for each query. To exploit common data retrievals and computation in concurrent queries, relational engines employ techniques ranging from constructing materialized views to optimizing multiple queries and sharing concurrent scans to the same table. These three techniques are briefly described next.

Materialized view selection [8] is typically applied to workloads known in advance, in order to speed up queries that contain common sub-expressions. Materialized views exploit commonality between different queries at the expense of potentially significant view maintenance costs. Tools for automatic selection of materialized views take such costs into account when recommending a set of views to create. The usefulness of materialized views is limited when the workload is not always known ahead of time or the workload requirements change frequently.

Multiple-query optimization (MQO) [10] identifies common sub-expressions in query execution plans during optimization, and produces globally-optimal

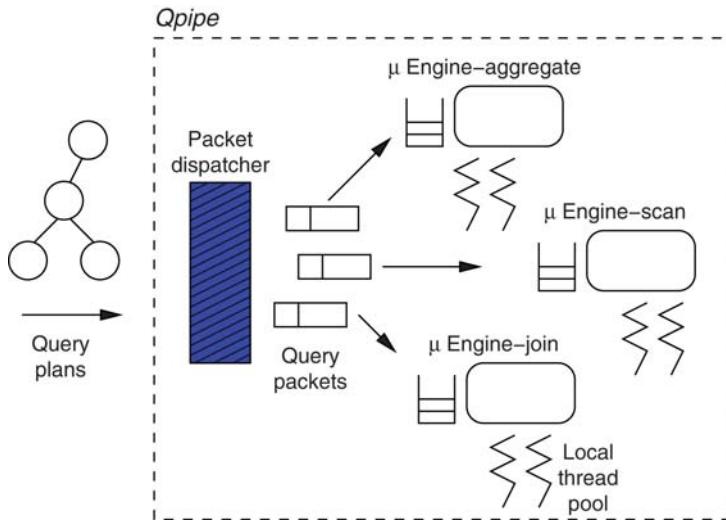
plans. The detection of common sub-expressions is performed at optimization time, thus, all queries need to be optimized as a batch. In addition, to share intermediate results among queries, MQO typically relies on costly materializations. To avoid unnecessary materializations, a study described in [9] introduces a model that decides at the optimization phase which result can be pipelined and which needs to be materialized to ensure continuous progress in the system.

Shared scans allow multiple independent concurrent scans to the same table on disk to be synchronized, so that each new page fetched from disk is consumed by all scans that include the page in their range. This optimization applies to scans that can receive their input pages in any arbitrary order. Since queries interact with the buffer pool manager through a page-level interface, it requires a certain engineering effort to develop generic policies to coordinate current and future accesses from different queries to the same disk pages. Several commercial DBMSs (Teradata, Microsoft's SQL Server, IBM's DB2 [6]) and research prototypes [4,14] incorporate various forms of multi-scan optimizations.

A relational engine based on the Staged DBMS design complements the above-mentioned techniques, by proactively coordinating same-operator execution among concurrent queries, thereby exploiting common accesses to memory and disks as well as common intermediate result computation. Such staged relational engines are the academic prototypes QPipe and Cordoba [4,5], developed at Carnegie Mellon University.

To maximize data and work sharing at execution time, QPipe monitors each relational operator for every active query in order to detect overlaps. For example, one query may have already sorted a file that another query is about to start sorting; by monitoring the sort operator QPipe can detect this overlap and reuse the sorted file. Once an overlapping computation is detected, the system executes the corresponding operation only once, and simultaneously pipelines the results of the common operation to the interested parties, thereby avoiding materialization costs.

QPipe follows a “one-operator, many-queries” design philosophy. Each relational operator is promoted to a staged, independent micro-engine which manages a set of threads and serves queries from a queue (see Fig. 1). A packet dispatcher converts an incoming query plan to a series of query packets. Data flow between micro-engines occurs through dedicated



Staged DBMS. Figure 1. QPipe architecture: queries with the same operator queue up at the same micro-engine (for simplicity, only three micro-engines are shown).

buffers - similar to a parallel database engine. QPipe optimizes resource utilization by grouping requests of the same nature together, and by having dedicated micro-engines to process each group of similar requests. Every time a new packet queues up in a micro-engine, all existing packets are checked for overlapping work. On a match, each micro-engine can employ different mechanisms for data and work sharing, depending on the enclosed relational operator. Such mechanisms are described in detail in [4]. Subsequent work on QPipe produced the Cordoba prototype which is suited for execution on multicore CPUs. The tradeoffs of work sharing in highly parallel multicore chip designs are discussed in [5].

Future Directions

By the year 2005 it was apparent to microprocessor designers that performance increases in next generation CPUs would come by incorporating an increasing number of CPU cores on the same chip. If this trend continues to hold, then software designers will need to devise efficient solutions to take advantage of the (increasing) hardware-available parallelism. A preliminary study of bottlenecks for database systems on multicore CPUs (chip multiprocessors or CMPs) that was published in 2007 can be found in [1]. A future direction for Staged database systems is to exploit their inherent parallelism nature and apply it to CMP designs.

From a software engineering point of view, years of DBMS software development have lead to complex

implementations that are increasingly difficult to extend, tune, and evolve. While software developers commonly organize code into separate components, the final product consists of tightly integrated and interdependent software modules, “glued” together to eliminate overheads and increase performance. It has been argued that such monolithic systems are “one size fits all” designs [11] that cannot possibly excel in all areas of data management. Another potential future direction for StagedDB is to allow several software components with specialized functionality to be transparently integrated and used inside a single system, thereby achieving high performance in several areas of data management without needing a number of different specialized architectures.

Cross-references

- [Operator-level Parallelism](#)
- [Parallel Database](#)

Recommended Reading

1. Hardavellas N., Pandis I., Johnson R., Mancheril N., Ailamaki A., and Falsafi B. Database servers on chip multiprocessors: limitations and opportunities. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007.
2. Harizopoulos S. Staged Database Systems. PhD Thesis, Computer Science Department, Carnegie Mellon University, 2005.
3. Harizopoulos S. and Ailamaki A. A case for staged database systems. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.

4. Harizopoulos S., Shkpenyuk V., and Ailamaki A. QPipe: a simultaneously pipelined relational query engine. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 383–394.
5. Johnson R., Hardavellas N., Pandis I., Mancheril N., Harizopoulos S., Sabirli K., Ailamaki A., and Falsafi B. To Share Or Not To Share? In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 351–362.
6. Lang C., Bhattacharjee B., Malkemus T., Padmanabhan S., and Wong K. Increasing buffer-locality for multiple relational table scans through grouping and throttling. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 1136–1145.
7. Larus J.R. and Parkes M. Using cohort-scheduling to enhance server performance. In Proc. General Track of the USENIX Annual Technical Conf., 2002, pp. 103–114.
8. Roussopoulos N. View indexing in relational databases. ACM Trans. Database Syst., 7(2):258–290, 1982.
9. Roy P., Seshadri S., Sudarshan S., and Bhobe S. Efficient and Extensible Algorithms for Multi Query Optimization. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 249–260.
10. Sellis T.K. Multiple query optimization. ACM Trans. Database Syst., 13(1):23–52, 1988.
11. Stonebraker M., Bear C., Cetintemel U., Cherniack M., Ge T., Hachem N., Harizopoulos S., Lifter J., Rogers J., and Zdonik S. One size fits all? - Part 2: Benchmarking results. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 173–184.
12. Stonebraker M., Held G., Wong E., and Kreps P. The design and implementation of Ingres. ACM Trans. Database Syst., 1(3):189–222, 1976.
13. Welsh M., Culler D., and Brewer E. Seda: an architecture for well-conditioned, scalable internet services. In Proc. 18th ACM Symp. on Operating System Principles, 2001, pp. 230–243.
14. Zukowski M., Héman S., Nes N., and Boncz P.A. Cooperative scans: dynamic bandwidth sharing in a DBMS. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 723–734.

A user query represents the user's information need. A user information need is hidden, depending on what the user already knows, what the user wants to find out about, and what is the users constraint (time, format, price, recent, location). Recall and precision are the basis of relevance-based effectiveness measures. Other commonly used evaluation measures based on recall and precision include the F-measure, the 11-point precision-recall curve, and the average precision.

Historical Background

The first formal evaluation of information retrieval (IR) systems was conducted in the late 1950s by Cyril Cleverdon at the College of Aeronautics in Cranfield, England [2]. These studies are often referred to as the Cranfield experiments and are the foundation of future research on IR evaluation. The Cranfield model requires an experimental setting with a test document collection, a set of queries, and relevance judgements that tell the relationships between the documents and the queries. Recall and precision are the effectiveness measures used in the Cranfield experiments, where the former is the percentage of the relevant documents that are retrieved and the latter is the percentage of the retrieved documents that are relevant.

TREC (Text REtrieval Conference) [3] is the most notable large IR evaluation project. The conference, co-sponsored by the National Institute of Standards and Technology (NIST) and the Advanced Research and Development Activity (ARDA) center of the U.S. Department of Defense, started in 1992 and convenes once a year. At the conference TREC participants from different research institutions evaluate and compare their IR systems on large test collections that consist of many gigabytes of documents. TREC usually uses the Cranfield model for evaluation setup and variations of recall and precision as effectiveness measures. The test collections and effectiveness measures used at TREC have been widely adopted by IR researchers as the standards for the evaluation of information retrieval systems.

Standard Effectiveness Measures

ETHAN ZHANG^{1,2}, YI ZHANG¹

¹University of California-Santa Cruz, Santa Cruz, CA, USA

²Yahoo! Inc., Santa Cruz, CA, USA

Synonyms

Evaluation in information retrieval

Definition

The standard effectiveness evaluation in information retrieval centers around determining how relevant are the documents retrieved to users' information needs.

Foundations

Experimental Setup

To evaluate an information retrieval (IR) system in the standard way, an experimental setting needs to be created that involves three things: i) a test document collection, ii) a set of information needs that are

represented as queries, and iii) relevance judgements for each query-document pair. The relevance judgement is made by one or more human assessors who determine whether a document is *relevant* or *irrelevant* to a query. A document is considered relevant to a query if the document satisfies the user information need represented by the query. After each retrieval experiment for a given query, four document sets are formed and counted: retrieved and relevant documents (tp), retrieved and irrelevant documents (fp), relevant and nonretrieved documents (fn), and irrelevant and nonretrieved documents (tn). It is clearer to illustrate the four document sets in the confusion matrix in [Table 1](#).

A basic assumption of the Cranfield model is that relevance judgement is available for each query-document pair in the collection. However, it is practically impossible to obtain exhaustive relevance judgements in an experiment with millions of documents, which is very common in information retrieval evaluations. In such cases, assessors usually only provide relevance judgements for the top- n results returned by several different search systems for each query. In some evaluation experiments, the retrieval systems perform a new search using new queries based on the judged documents and newly retrieved documents in the top- n results are then judged. This process is iterated until no new relevant documents are found in the top- n results.

Recall and Precision

Recall and precision are the basis of relevance-based retrieval effectiveness measures. They have been the most commonly used IR evaluation measures since the Cranfield experiments, and are considered the “gold standard” of IR evaluation by many researchers. Recall is the fraction of all relevant documents that have been retrieved by an IR system. Precision is the fraction of all retrieved documents that are relevant. In terms of the numbers in the confusion matrix, recall is

$$R = \frac{tp}{tp + fn}$$

and precision is

$$P = \frac{tp}{tp + fp}$$

Recall and precision trade off against each other. One can easily build a system with a recall value of one by returning all documents in the collection, but the precision in this case would be very low. In contrast, a search engine can only return a few documents that some user has judged to be relevant, which would possibly result in high precision but have low recall.

The fact that two numbers are used to evaluate an IR system allows researchers to emphasize on one measure versus the other in various circumstances. For web search engines, a typical user would like the first few search results to be relevant (high precision), however he/she usually does not have enough time to read every document that is relevant. On the other hand, a research scientist surveying a research topic would like to see every relevant document (high recall) while tolerating low precision to a certain degree.

F-Measure

The *F-measure* is a single number that combines recall and precision. The measure is formally defined as the weighted harmonic mean of recall and precision, or

$$F = \frac{1}{\alpha \frac{1}{P} + (1 - \alpha) \frac{1}{R}}$$

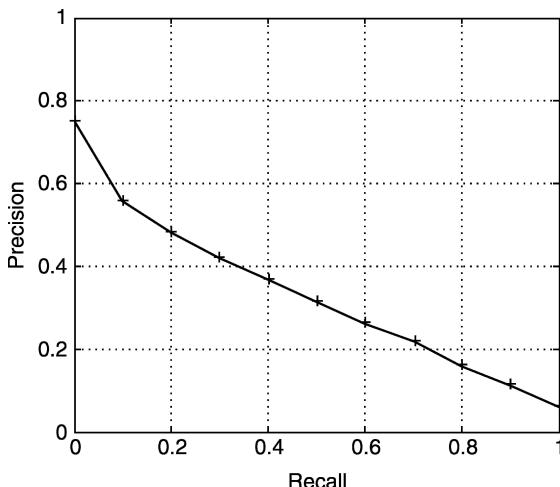
where $\alpha \in [0,1]$ is used to adjust the weighting of R and P . F is equivalent to recall when $\alpha = 0$ and precision when $\alpha = 1$. The commonly used F-measure has $\alpha = 1/2$ and weighs recall and precision equally. It can be written as $F_1 = \frac{2PR}{P+R}$. This measure can be viewed as a compromise between recall and precision. It is high only when both recall and precision are high.

Precision-Recall Curve

Recall and precision treat retrieved documents as a set, in which the order of the documents does not matter. However, most modern search systems return a ranked list of documents, where a document is ranked higher if the system believes it is more likely to be relevant to the query. For such systems, the *precision-recall curve* is a commonly used tool to evaluate the retrieval performance. The precision-recall curve plots precision as a function of recall. Standardly the precision is

Standard Effectiveness Measures. Table 1. Confusion matrix, where tp , fp , fn and tn correspond to the number of documents that fall into the corresponding category

	Relevant	Irrelevant
Retrieved	true positive (tp)	false positive (fp)
Not retrieved	false negative (fn)	true negative (tn)



Standard Effectiveness Measures. Figure 1. A 11-point precision-recall curve.

interpolated and plotted at 11 recall levels, $r = 0.0, 0.1, \dots, 1.0$. For multiple queries, the precisions at the same recall level can be averaged over all queries. Figure 1 shows the 11-point precision-recall curve of one of the best systems in TREC 8.

Average Precision

Average Precision (AP), which has become a standard effectiveness measure among the TREC community in recent years, is a combination of recall and precision for ranked retrieval results. Given a ranked list of documents for a single query, the average precision is the mean of the precision scores after each relevant document is retrieved.

$$\text{Average Precision} = \frac{\sum_r P@r}{R}$$

where r is the rank of each relevant document, R is the total number of relevant documents, and $P@r$ is the precision of the top- r retrieved documents. This measure is very sensitive to the rankings of the relevant document in the retrieval results, and therefore a good measure for tuning ranking algorithms. For one query, the average precision is approximately the area under the uninterpolated precision-recall curve.

ROC Curve

An alternative to the precision-recall curve is the *Receiver Operating Characteristics* (ROC) curve, which plots true positive rate (*sensitivity*) against false positive rate (*1-specificity*). In this setting, *sensitivity*

is a synonym for recall, and false positive rate is $fp/(fp + tn)$ (therefore specificity is $tn/(fp + tn)$).

Key Applications

Retrieval effectiveness evaluation is an integral part of designing an IR system or algorithm. The standard effectiveness measures are used to compare a new system with benchmark systems and to justify the value of the new system. Although the relevance-based evaluation model has drawn criticism since the beginning, it is still the most widely accepted and used approach for IR evaluation. It is fair to say that the standard measures and their variations are used wherever information retrieval algorithms are built.

Data Sets

TREC has by far the largest data collections for information retrieval evaluation. The most often used collections are those that were built for the Ad Hoc retrieval track in the first 8 TREC conferences. In total these collections consist of 1.89 million documents. For each conference, TREC also collected relevance judgements for 50–100 information needs, which are called topics in TREC. There are relevance judgements for totally 450 information needs. Other TREC collections used for information retrieval evaluations are: TREC Web track collections, TREC terabyte track collections, TREC Blog Track collections, TREC Enterprise Track collections, TREC Filtering Track collections, TREC Genomics Track collections, TREC HARD Track collections, TREC Interactive Track collections, TREC Legal Track collections, TREC Novelty Track collections, TREC Robust Track collections, TREC Query Track collections, TREC Question Answering Track collections, and TREC SPAM Track collections. For details about the collections and different standard evaluation measures used for each collection, readers may refer to book [3] or visit TREC's web site at <http://trec.nist.gov>.

Cross-references

- ▶ Average Precision
- ▶ Effectiveness Involving Multiple Queries
- ▶ Eleven Point Precision-recall Curve
- ▶ F-Measure
- ▶ Information Retrieval
- ▶ Precision
- ▶ Precision at n
- ▶ Precision-Oriented Effectiveness Measures
- ▶ Recall

Recommended Reading

1. Baeza-Yates R. and Ribeiro-Neto B. Modern Information Retrieval. Addison Wesley, New York, NY, USA, 1999.
2. Cleverdon C.W. The significance of the Cranfield tests on index languages. In Proc. 14th Annual Int. Conf. on Research and Development in Information Retrieval, 1991, pp. 3–12.
3. Voorhees E.M. and Harman D.K. (eds.). TREC: Experiment and Evaluation in Information Retrieval. MIT, Cambridge, MA, USA, 2005.

Standing Query

- ▶ Continuous Query

Star Index

THEODORE JOHNSON
AT&T Labs Research, Florham Park, NJ, USA

Synonyms

[Star index](#); [Join index](#); [Join indices](#)

Definition

A star index is a collection of join indices, one for every foreign key join in a star or snowflake schema.

Key Points

A common structure for a data warehouse is a fact table consisting of several *dimension* fields and several *measure* fields. To reduce storage costs, the fact table is often normalized into a *star* or a *snowflake* schema. Since most queries reference both the (normalized) fact tables and the dimension tables, creating a star index can be an effective way to accelerate data warehouse queries. The Red Brick data warehouse system has implemented star indices.

Cross-references

- ▶ [Join Index](#)
- ▶ [Star Schema](#)
- ▶ [Snowflake Schema](#)

Star Join Schema

- ▶ [Star Schema](#)

Star Schema

KONSTANTINOS MORFONIOS, YANNIS IOANNIDIS
University of Athens, Athens, Greece

Synonyms

[Star join schema](#)

Definition

A *star schema* has one “central” table whose primary key is compound, i.e., consisting of multiple attributes. Each one of these attributes is a foreign key to one of the remaining tables. Such a foreign key dependency exists for each one of these tables, while there are no other foreign keys anywhere in the schema. (In the above, without loss of generality, the assumption is made that all these other tables have simple primary keys. This is usually the case in almost all practical situations, as for efficiency, these keys are typically generated *surrogate keys*.)

Key Points

Most data warehouses that represent the multidimensional conceptual data model in a relational fashion [1,2] store their primary data as well as the data cubes derived from it in star schemas. The “central” table and the remaining tables of the definition above correspond, respectively, to the *fact table* and the *dimension tables* that are typically found in data warehouses. Each *fact* (tuple) in the fact table consists of a set of numeric *measures*, comprising the objects of analysis, and a set of *dimensions*, which uniquely determine the set of measures. The dimension tables are usually smaller than the fact table and store the attributes of the aforementioned dimensions.

For example, consider a data warehouse of a retail chain with many stores around a country. The dimensions may be the products sold, the stores themselves with their locations, and the dates, while the numeric measures may be the number of items and the total monetary amount corresponding to a particular product sold in a particular store on a particular date. The relevant star schema is shown below, where *SalesSummary* is the fact table, primary keys are in Italic, and each attribute of the fact-table primary key is a foreign key to one of the other tables.

SalesSummary(*ProductId*, *StoreId*, *DateId*,
NumOfItems, *TotalAmount*)

Product(*ProductId*, *ProdName*, *ProdDescr*, *Category*, *CategoryDescr*, *UnitPrice*)

Store(*StoreId*, *Street*, *City*, *State*)

Date(*DateId*, *Day*, *Month*, *Year*)

If one were to draw the above as a graph, with tables as nodes and foreign keys as edges, or even as an ER diagram, with the fact table as a relationship and the dimension tables as entities, the resulting image is that of a star, with the fact table in the middle, hence, the name of these schemas.

Finally, note that dimensions often consist of several attributes organized in hierarchies. For instance, dimension *Store* in the example above contains values at different levels of detail, forming the hierarchy *Street*→*City*→*State*. As also shown in the example, star schemas capture all levels of a hierarchical dimension in a single, non-normalized table. An extension of the star schema that explicitly captures hierarchies in the dimensions is the *snowflake schema*.

Cross-references

- ▶ [Cube Implementations](#)
- ▶ [Data Warehouse](#)
- ▶ [Dimension](#)
- ▶ [Hierarchy](#)
- ▶ [Measure](#)
- ▶ [Multidimensional Modeling](#)
- ▶ [Snowflake Schema](#)

Recommended Reading

1. Chaudhuri S. and Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Rec., 26(1):65–74, 1997.
2. Kimball R. and Ross M. The data warehouse toolkit: The complete guide to dimensional modeling. Wiley, New York, NY, USA, 2nd edn., 2002.

Star Schema Modeling

- ▶ [Multidimensional Modeling](#)

State Query

- ▶ [Timeslice Operator](#)

State-based Publish/Subscribe

HANS-ARNO JACOBSEN

University of Toronto, Toronto, ON, Canada

Definition

State-based publish/subscribe is an instance of the publish/subscribe concept. However, it is distinguished from other publish/subscribe approaches by maintaining partial matching state when processing publications, whereas, traditionally, publish/subscribe treat publications as transient and does not manage matching state. State-based publish/subscribe support the detection of composite events, event correlation and complex event processing.

Key Points

In terms of publishing, subscribing, and decoupling, state-based publish/subscribe is no different from topic-based or content-based publish/subscribe. The main difference to the other publish/subscribe approaches is that state-based publish/subscribe treats publications as non-transient. A publication is processed by the publish/subscribe system and builds up partial matching state, contributes to existing partial matching state, triggers notifications if a match is complete, or is discarded, if no matching subscription exists. This is unlike in the other publish/subscribe approaches that treat publications as transient messages, where the arriving publication is matched against the subscriptions stored with the system and forwarded towards all matching subscribers, or dropped, if no matching subscription exists. In state-based publish/subscribe, the publish/subscribe system carries state across the processing of different publications. That is, various different publications arriving over time are correlated based on conditions expressed in subscriptions to result in matching subscriptions.

The publication data model is exactly the same as in a topic-based or a content-based publish/subscribe model, depending on the nature of the state-based approach, either topic-based, or content-based.

The subscription language model follows suite, but greatly extends the subscription language capabilities with means to express the correlation of publications. These capabilities are added to allow the application to express so called *composite subscriptions*. A composite subscription is the combination of several individual

atomic subscriptions by means of an operator algebra that allows the developer to compose individual atomic subscriptions. An atomic subscription is a subscription in the publish/subscribe sense. It is referred to as atomic because it is matched by a single publication. A composite subscription defines a *composite event*. A composite event defines the set of events that have to occur in the specified constellation in order for the composite subscription to match. In publish/subscribe, the notion of event and publication are synonymous, while only the term composite event is used. Also, the term composite event is often used to refer to the composite subscription expression without differentiating between event and subscription.

There are large differences in the expressive power of subscription languages for specifying composite subscriptions. Common operators include the specification of composite-and (all specified events must occur in any order), composite-or (one of the events must occur), sequence operator (the specified sequence of events must occur and other events may or may not be interspersed with the sequence), and regular expression pattern (the specified pattern of events must occur). In addition, reference to time is included in many subscription languages to delimit the time a composite subscription can remain in a partial matching state before resetting to a completely unmatched state. More generally, various consumption policies attached to subscriptions express what should happen with partial matching state, as it accumulates in the system and new events arrive. For example, a consumption policy could express that a newly arriving event is correlated with the oldest or the newest event in a composite event that has accumulated a partial matching state holding many individual events already. Consumption policies are a powerful way to customize the matching behavior of state-based publish/subscribe systems.

In the state-based publish/subscribe model, the publish/subscribe matching problem is defined as follows: Given a set of subscriptions, S , and a sequence of events, E , as seen by the publish/subscribe system, determine the subscriptions in S that match under E . This formulation of the matching problem is different from the standard publish/subscribe matching problem that only looks at a single event e at a time. In state-based publish/subscribe, the matching algorithm has to manage partial matching state and correlate newly arriving events with already existing partial

matching state stored in the algorithm's data structures.

State-based publish/subscribe is a fairly new sub-classification of publish/subscribe, consequently few established standards and products exist that refer to this model. However, several research projects are experimenting with the above described schemes. For example, the PADRES [?,?] project is an example of a state-based publish/subscribe system. Moreover, many rule-based engine products, such as JESS and Drools are similar in conception to the above described functionality. There is a large product space of older and emerging rule-based systems that are applied to similar applications as state-based publish/subscribe. There is no clear dominant player at this point. The key difference between these approaches and the here described state-based publish/subscribe model, is the notion of a rule, which essentially is a composite subscription; except that rules follow a stricter if-then-else syntax model than composite subscriptions, which often only capture the antecedent part of a rule. Moreover, there is an emerging space of complex event processing, event correlation, or simply correlation technology, which also falls under the here defined space of state-based publish/subscribe systems. State-based publish/subscribe targets applications that need correlate events over time. Applications of this nature are event correlation for network management, system management and diagnostics, and business process execution and business activity monitoring. In these application scenarios, large numbers of events that in isolation are not useful, need to be correlated to detect higher-level composite events.

In the literature, the term state-based publish/subscribe is not used uniformly. Also, state-based publish/subscribe is just emerging as a separate model. The functionality provided by a state-based publish/subscribe system is very close to what rule-based systems offer. However, unlike rule-based systems, state-based publish/subscribe does not explicitly use rules to let developers model applications. Besides capturing state as the correlation of events, the management of publication state and subscription state are another important property of publish/subscribe systems, which falls into the subject spaces publish/subscribe model.

Cross-references

- ▶ [Publish/Subscribe](#)
- ▶ [Subject Spaces](#)

Recommended Reading

1. Fidler E., Jacobsen H.-A., Li G., and Mankovski S. The PADRES distributed publish/subscribe system. In Feature Extractions in Telecom. and Softw. Syst., S. Reiff-Marganie, and M. Ryan (eds.), IOS Press, 2005.
2. Li G., and Jacobsen H.-A. Composite subscriptions in content-based publish/subscribe systems. In Proc. ACM/IFIP/USENIX 6th Int. Middeware Conf., 2005.

evaluation was to minimize redundancy of computation when answering a user query. A second important consideration in designing statistical DBMS is security so that the individual data records are not exposed to a malicious user. Several strategies are used to attain this security including (i) allowing the user to ask only aggregate queries, (ii) answering a query with a range of values instead of exact values, (iii) disallowing a user to make repeated increasingly specific queries that might expose actual data values.

Statistical Correctness

- ▶ [Summarizability](#)

Statistical Data Management

AMARNATH GUPTA

University of California-San Diego, La Jolla, CA, USA

Synonyms

[Statistical database](#)

Definition

A Statistical data management system is a data management system designed to explicitly handle so called “macro data,” i.e., data computed by different forms of summarization, including grouping and classification, as first class objects. In a statistical data management system, there are data manipulation operators that “slice and dice” the macro data. In many statistical databases, one of the goals is to hide the micro data (i.e., the raw data records from which the macro data are computed) from user queries.

Example: A classical example of a statistical database is a database created for social or economic surveys. STORM [4] is a classical statistical data management system.

Key Points

Research on statistical databases started in the 1970s and flourished in the 1980s, predating OLAP. A number of systems developed based on both relational and object-oriented data models. Many of these systems developed a graph-based representation of statistical data where one could construct hierarchies of categories using different attributes, and place the summarized data under the suitable categories. One goal of query

Cross-references

- ▶ [Privacy](#)
- ▶ [On-line Analytical Processing](#)
- ▶ [Secure Database Development](#)
- ▶ [Summarizability](#)

Recommended Reading

1. Denning D.E. and Schlörer J. A fast procedure for finding a tracker in a statistical database. ACM Trans. Database Syst., 5(1):88–102, 1980.
2. Ghosh S.P. Statistical relational tables for statistical database management. IEEE Trans. Softw. Eng., 12(12):1106–1116, 1986.
3. Rafanelli M. and Ricci F.L. Mefisto: a functional model for statistical entities. IEEE Trans. Knowl. Data Eng., 5(4):670–681, 1993.
4. Rafanelli M. and Shoshani A. STORM: a statistical object representation model. In Proc. 2nd Int. Conf. on Scientific and Statistical Database Management, 1990, pp. 14–29.
5. Shoshani A. OLAP and statistical databases: similarities and differences. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 185–196.

Statistical Database

- ▶ [Statistical Data Management](#)

Statistical Decision Techniques

- ▶ [Classification](#)

Statistical Disclosure Control (SDC)

- ▶ [Inference Control in Statistical Databases](#)

Statistical Disclosure Limitation (SDL)

► Inference Control in Statistical Databases

Statistical Disclosure Limitation For Data Access

STEPHEN E. FIENBERG, JIASHUN JIN
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

Confidentiality protection; Multiplicity; Privacy protection; Restricted data; Risk-utility tradeoff

Definition

Statistical Disclosure Limitation refers to the broad array of methods used to protect confidentiality of statistical data, i.e., fulfilling an obligation to data providers or respondents not to transmit their information to an unauthorized party. *Data Access* refers to complementary obligations of statistical agencies and others to provide information for statistical purposes without violating promises of confidentiality.

Historical Background

Starting in the early twentieth century, U.S. government statistical agencies worked to develop approaches for the protection of the confidentiality of data gathered on individuals and organizations. As such agencies also have a public obligation to use the data for the public good, they have developed both a culture of confidentiality protection and a set of statistical techniques to assure that data are released in a form that limits the identification of individual data providers [2]. In a now classic 1977 paper, Dalenius [3] described the probabilistic notion of a disclosure: “If the release of the statistics S makes it possible to determine the value [of confidential statistical data] more accurately than is possible without access to S , a disclosure has taken place.” The ensuing statistical literature on disclosure limitation has built on this probabilistic notion.

Foundations

Privacy, Confidentiality, and Individual Identification

Massive databases and widespread data collection and processing offer enormous opportunities for statistical

analyses, advances in the understanding of social and health problems, and benefits to society more broadly. But the explosion of computerized databases containing financial and healthcare records, and the vulnerability of databases accessible via the Internet, has heightened public attention and generated fears regarding the privacy of personal data. Identify theft and sensitive data disclosure may be just a click away from a new generation of computer users and potential intruders.

Data collected directly under government auspices or at public expense are in essence a public good; legitimate analysts wish to utilize the information available in such databases for statistical purposes. Thus, society’s challenge is how to release the maximal amount of information without undue risk of disclosure of individually identifiable information. Assessing this tradeoff is inherently a statistical matter, as is the development of methods to limit disclosure risk. What distinguishes the field of statistical disclosure limitation from many other approaches to privacy protection is the ultimate goal of data access and enhanced data utility.

The term *privacy* is used both in ordinary language and in legal contexts with a multiplicity of meanings. Among these is the concept of privacy as “the right to be let alone,” e.g., see Warren and Brandeis [14], and privacy in the context of data as the control over information about oneself. But privacy is personal and subjective, varies from one person to another, and varies with time and occasion depending on the context. It is even more difficult to define precisely the meaning of “privacy-preserving” with respect to databases, and the data pertaining to individual entities contained therein.

Confidentiality is the agreement, explicit or implicit, between data subject and data collector regarding the extent to which access by others to personal information is allowed. Confidentiality protection has meaning only when the data collector can deliver on its promise to the data provider or respondent. Confidentiality can be accorded to both individuals and organizations; for the individual, it is rooted in the right to privacy (i.e., the right of individuals to control the dissemination of information about themselves), whereas for establishments and organizations, there are more limited rights to protection, e.g., in connection with commercial secrets.

Disclosure relates to inappropriate attribution of information to a data provider or intruder, whether

to an individual or organization. There are basically two types of disclosure, identity and attribute. An identity disclosure occurs if the data provider is identifiable from the data release. An attribute disclosure occurs when the released data make it possible to infer the characteristics of an individual data provider more accurately than would have otherwise been possible. The usual way to achieve attribute disclosure is through identity disclosure; first one identifies an individual through some combination of variables and then one associates with that individual values of other variables included in the released data.

Statistical disclosure limitation (SDL) is a set of techniques designed to “limit” the extent to which databases can be used to glean identifiable information about individuals or organizations. The dual goals of SDL are to assure that, based on released data, respondents can be identified only with relatively low probability, but also to release data that are suitable for non-identifiable analytical statistical purposes.

The Intruder

To protect the confidentiality of statistical data, one needs to understand what intruders or data snoopers want and how they may learn information about individuals in a database that require protection. Intruders may be those with legitimate access to databases and/or those who gain access to a database by breaking security measures designed to keep them out. In either case, one needs to distinguish among

- *Intruders with a specific target*, e.g., a friend or relative. The intruder may already know that the respondent is included in the database, will possess information about the target (e.g., height, weight, habits, income) and will search the database in order to learn additional information, e.g., drug and alcohol use.
- *Intruders in possession of data on multiple individuals whose goal is record linkage*, e.g., to build a larger database containing more individual information. Data consolidators or aggregators fit within this category.
- *Intruders without any specific target, whose goal is to embarrass the data owner*. The intruder may be an enemy agent or a “hacker” eager to demonstrate a capability of breaking through efforts to limit disclosure.

Data owners can be successful in protecting the confidentiality of released data if the intruder remains

sufficiently uncertain about a protected target value after data release. Various authors in the SDL literature discuss confidentiality protection from the perspective of protecting against intruders or data snoopers, e.g., see [8,12], especially those using record linkage methods [11] for attempting to identify individuals in databases.

One may consider the intruder as someone engaged in a form of a large number of statistical tests, each at significance level α associated with an effort to identify an individual in the database. The data owner needs to account for this somehow. Some of the null hypotheses will eventually be rejected whether or not they are actually false, and thus there is a problem for the intruder as well. For the data owner, simply controlling the probability of erroneously identifying each respondent, at say 1%, is not enough. To ensure that the probability that at most one out of 1,000 individuals in a sample will be identified to be less than 1%, one in fact needs to assure that the probability of identifying *each* individual is no greater than $\approx 1\% / 1000 = 10^{-5}$.

Statistical Analysis Methods for Protecting Privacy

Matrix Masking refers to a class of SDL methods used to protect confidentiality of statistical data, transforming an $n \times p$ (cases by variables) data matrix Z through pre- and post-multiplication and the possible addition of noise. The four most common forms of masking are:

1. *Sampling* clearly provides a measure of direct protection from disclosure provided that there is no information of which individuals or units are included in the sample. An intruder wishing to identify an individual in the sample and link that person's information to data in external files, using “key” variables such as age and geography available in both databases, needs to determine whether a record is unique in the sample, and if so, the extent to which a record that is unique in the sample is also unique in the population. For continuous variables, virtually all individuals are unique in the sample, and one needs to understand the probability that an intruder would correctly match records, e.g., in the presence of error in the key variables (e.g., see Fienberg et al. [8]). For categorical data, uniqueness corresponds to counts of “1” and various authors have shown, roughly speaking, that the probability that

an individual record that is unique in the sample is also unique in the population from which the sample was drawn equals the sampling fraction, n/N , e.g., see [7]. Thus for a sample of size 2,000 drawn from a population of 200,000,000 adults the sampling fraction is $2,000/200,000,000$ or 0.00001. The bottom line therefore is that sampling protects, just not absolutely.

2. *Perturbation* is an approach to data masking in which the transformation involves random perturbations of the original data, either through the addition of noise or via some form of restricted randomization. The simplest form of perturbation is the addition of noise. Common forms for the noise are observations drawn from a normal distribution with zero mean or perhaps a double exponential, also centered at zero. Someone analyzing the resulting transformed data must statistically reverse the noise addition process using methods from the literature on measurement error models – this requires release of the parameters of the noise component, e.g., the error variance in the normal case. Other examples of perturbation include data swapping and related tabular adjustment approaches, e.g., see [9].
3. *Collapsing* is also referred to using the labels micro-aggregation and global recoding in the statistical literature [15], and k -anonymity in the computer science literature. In the statistical literature on tabular categorical data, collapsing across variables in a table produces a marginal table and a popular form of data release to protect confidentiality is the release of multiple marginal tables, especially when they correspond to the minimal sufficient statistics of a log-linear model. For more details, see [10].
4. *Synthetic data* are used to replace a database by a similar one, for which the individuals are generated through some statistical process. This can be achieved through the repeated application of data swapping, e.g., see [9], or the method known as multiple imputation, e.g., [13].

Implicit in all of these techniques is the notion that when masked data are released they can be used by responsible analysts to carry out statistical analyses so that they can reach conclusions similar to those that they would have reached had they analyzed the original data. This means that all of the details of the transformation, both stochastic and non-stochastic, must be

made available to the user, a point not well understood in the computer science literature or by many statistical agencies. See the related discussion in [6]. Even when one has applied a mask to a data set, the possibilities of both identity and attribute disclosure remain, although the risks may be substantially diminished. Thus, one must still assess the extent of risk posed by the transformed data.

Putting SDL Methods to Use: Risk-Utility Tradeoff

If one is adding noise to a set of observations in order to protect confidentiality, how much noise is sufficient? And can one add too much? Clearly, too much noise will distort the data substantially and even if the details of the error variance are released the masking may impede legitimate statistical analyses of the data. The same is true for any of the methods of SDL. Thus, one faces a tradeoff between data protection and data utility, something that one can assess formally using statistical decision analysis and depict graphically, e.g., see the chapter by Duncan et al. in [5]. For a slightly less formal approach to the tradeoff for categorical data protection through the release of multiple margins, see [10].

A crucial but relatively rarely discussed aspect of the risk-utility tradeoff involves the issue of multiplicity, introduced above. For illustration, consider a data set with information on 2,000 individuals, for each of which records the diagnostic result of an HIV test, with 1 corresponding to a positive result and 0 to a negative one. To protect the confidentiality for those individuals with positive HIV test outcomes, the data owner adds noise to each record value.

Suppose an intruder wishes to identify the individuals corresponding to the proportion ε of “1”s ($0 < \varepsilon < 1$) and the data owner attempts to protect the records by adding i.i.d. Gaussian noise $N(0, \sigma^2)$ to each data point. Clearly σ needs to be large enough to disguise some of the 1s and make them hard to distinguish from some of the “0”s. Suppose that the intruder wants to make sure that most of those identified as “1”s are indeed “1”s (otherwise the attack on the database would be unsuccessful). In statistical terms, this means that the intruder must control for the False Discover Rate (FDR), i.e., the rate of misclassified “1”s out of all those individuals labeled as being “1”s [2,1]: $FDR = [\#\{\text{Misclassified “1”s}\}] / [\#\{\text{All classified “1”s}\}]$. Consider an intruder who decides

to set the FDR at 5% by picking a threshold σt and classifying any entry as a “1” if the observed value exceeds the threshold. By elementary statistics, the number of misclassified “1’s is distributed as a binomial random variable, $B(n(1 - \varepsilon), \bar{\Phi}(t))$, and the number of correctly classified “1’s is distributed as $B(ne, \bar{\Phi}(1 - \frac{1}{\sigma}))$. Consequently, the associated $FDR = \frac{n(1-\varepsilon)\bar{\Phi}(t)}{n(1-\varepsilon)\bar{\Phi}(t) + ne\bar{\Phi}(1-\frac{1}{\sigma})} \approx \left[1 + \left(\frac{\varepsilon}{1-\varepsilon}\right)\left(\frac{\bar{\Phi}(t-\frac{1}{\sigma})}{\bar{\Phi}(t)}\right)\right]^{-1}$, where $\bar{\Phi} = 1 - \Phi$ is the survival function of the standard normal distribution function.

Consider a high risk population where 50% of the individuals test positive for HIV, i.e., $\varepsilon = 1/2$ and suppose that the data owner chose $\sigma = 1$ as the noise variance to protect the data. To ensure that $FDR \leq 5\%$, the intruder needs to set $\left(\frac{\varepsilon}{1-\varepsilon}\right)\left(\frac{\bar{\Phi}(t-\frac{1}{\sigma})}{\bar{\Phi}(t)}\right) = 19$, which yields $t \approx 3.132$. The threshold is high enough so that the chance for each of the individuals exhibiting a “1” to be correctly classified as “1” is $\bar{\Phi}(t - 1) = \bar{\Phi}(2.132) \approx 0.0165$, which seems not very large. But since $n = 2000$, the number of “0’s that are misclassified as “1’s is approximately $n(1 - \varepsilon)\bar{\Phi}(3.132) \approx 2000 \times \frac{1}{2} \times 0.00087 = 0.87$, and the number of “1’s that are correctly classified as “1’s is approximately $ne\bar{\Phi}(2.132) = 2000 \times \frac{1}{2} \times 0.0165 \approx 16.5$. This says that the intruder is able to identify 17 records, out of which 16 are correctly classified!

Alternatively, one might ask about the probability that no more than k “1’s are correctly classified, i.e., $\sum_{j=0}^k \binom{ne}{j} p^j (1-p)^{ne-j}$, $p \equiv \bar{\Phi}(2.132)$. For $k = 0, 3, 6, 9$, the probabilities are correspondingly 5.45×10^{-8} , 5.22×10^{-5} , 2.62×10^{-3} , and 0.031. To understand the implications of these values, consider $k = 9$. This says that with probability as high as 97%, 9 or more records that are actually “1’s are correctly identified as “1”! For many this might seem to be a worrisome situation, and it raises issues associated with the efficacy of adding noise that have not appeared in the statistical literature on confidentiality protection.

Suppose that the data come from a low risk population where only 5% or 100 individuals test positive for HIV, i.e., $\varepsilon = 0.05$, and the data owner uses a similar level of noise addition for confidentiality protection, i.e., $\sigma = 1$. Then to ensure that $FDR \leq 5\%$, the intruder needs to set $\left(\frac{\bar{\Phi}(t-\frac{1}{\sigma})}{\bar{\Phi}(t)}\right) = 361$, which yields $t \approx 6.22$. Correspondingly, $\bar{\Phi}(t) = 2.5 \times 10^{-10}$

and $\bar{\Phi}(t - \frac{1}{\sigma}) = 8.9 \times 10^{-8}$. The expected number of “0” that are misclassified as “1” is approximately $n(1 - \varepsilon)\bar{\Phi}(6.22) = 2000 \times 0.95 \times 2.49 \times 10^{-10} \approx 4.7 \times 10^{-6}$, and the number of “1” that are correctly classified as “1” is approximately $ne\bar{\Phi}(5.22) = 2000 \times 0.05 \times 8.95 \times 10^{-8} \approx 8.95 \times 10^{-6}$. In this case, since $n\bar{\Phi}(6.22) \ll 1$, the approximation is inaccurate and one needs to take a different approach.

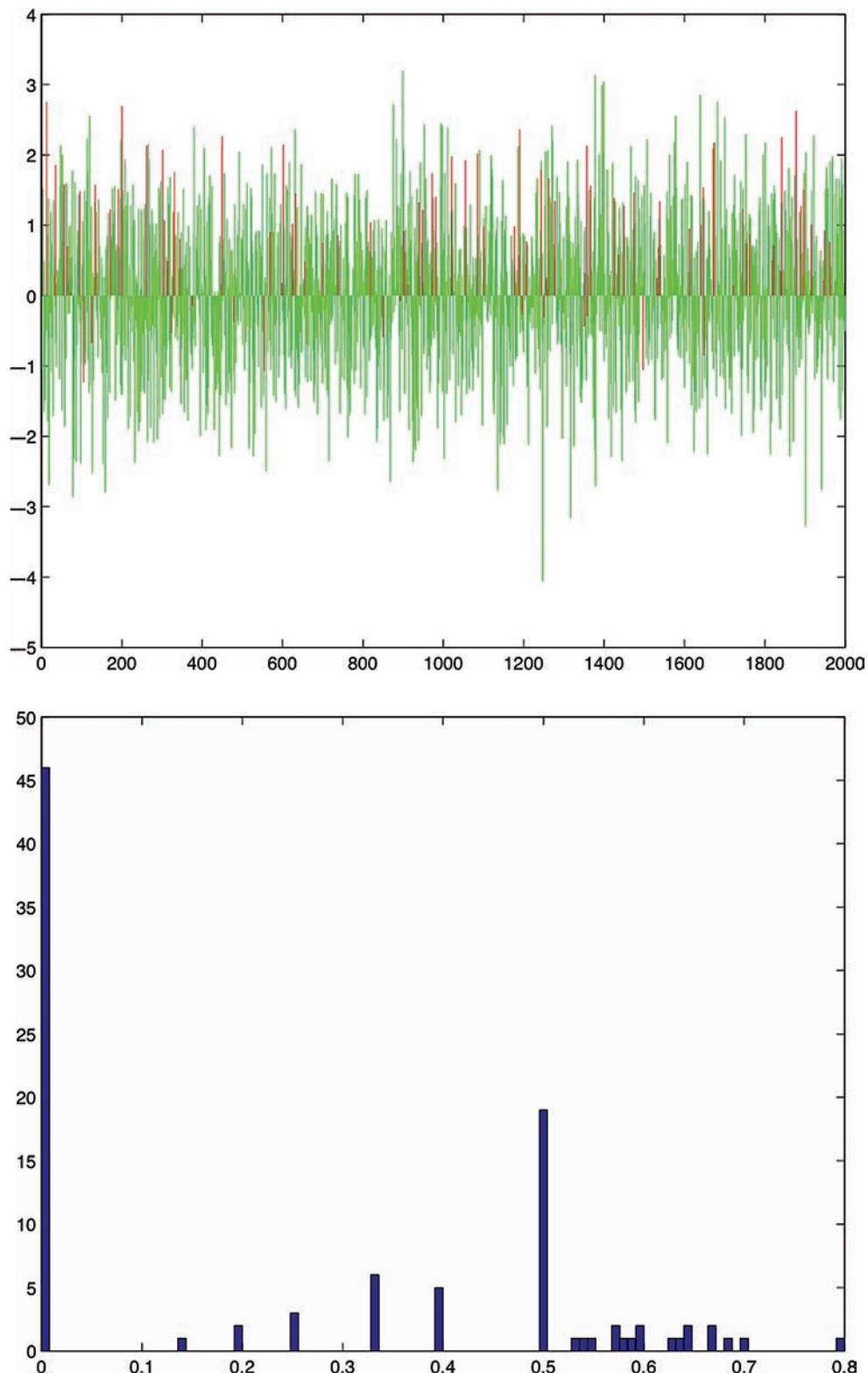
In fact, the proportion of true HIV cases is so small that the example falls into the so-called *very sparse regime* studied in detail in the multiple testing literature, see for example [1,4]. One phenomenon from that literature implies that when the noise level is relatively high, the extreme values are not necessary related to cases with positive HIV tests. Consider the following simulated data set with $n = 2,000$ cases, where 100 of them are HIV (equal to 1) and all others are non-HIV (equal to 0). The data owner adds independent standard Gaussian $N(0,1)$ noise to each value. Figure 1 shows the result where red correspond to cases with positive HIV tests, and green correspond to cases with negative HIV tests. The red values are larger than typical green ones, but not larger than all of them. In fact, among the largest 10 values, only 2 are red, with 8 are green.

This leads us to another interesting phenomenon from the statistical literature on the FDR. Let mFDR denote the minimum FDR across all possible thresholds t , $mFDR = \min_{\{t\}} \{FDR_t : FDR \text{ at the threshold } t\}$. How small can mFDR be? Figure 1 shows the histogram of the mFDR values for 100 independent repetitions of the simulation experiment. More than half of the time, the mFDR value is no less than 15%, and sometimes it is as great as 50% and larger!

This simple example implies that with $\sigma = 1$, the noise level might be so large that the intruder cannot correctly identify any HIV cases. But from the perspective of the risk-utility tradeoff, one also needs to ask whether the noise level is so high that the data are no longer analytically useful. Thus one needs to ask: What is the largest noise variance that still allows for valid inferences, c.f., [1,4]. If the number of true HIV cases is

$$m = m_n = n^{1-\beta}, \quad (1)$$

and the noise level is $\sigma = \sigma_n = \frac{1}{\sqrt{2r \log n}}$, where $0 < \beta$, $r < 1$ are parameters, then as n tends to ∞ , there is a boundary, $r = \beta$, which separates the β - r plane into two



Statistical Disclosure Limitation For Data Access. Figure 1. Left Panel: Perturbed HIV data through addition of independent draws from $N(0,1)$. Those values associated with positive HIV tests are in red, and those with negative HIV tests are in green. Right Panel: 100 simulated mFDR values based on 100 simulation for $n = 2,000$ and $\varepsilon = .05$ and added noise from $N(0,1)$.

regions: the *classifiable* region and the *non-classifiable* region; In the interior of the classifiable region, asymptotically, it is possible to isolate completely the cases with positive HIV tests from those with negative ones. In fact, there is a threshold by which one can identify that subset of the data corresponding to positive HIV tests. On the one hand, almost every “identified” HIV case has a positive HIV test and the subset includes almost all the cases with positive HIV tests. In the interior of the non-classifiable region, by contrast, such isolation of cases is impossible. In fact, given any chosen threshold, either one situation or the other occurs!

For the example of $n = 2,000$ and $m = 100$ cases with positive HIV tests, take $\beta = 1 - \log(m)/\log(n) \approx 0.3941$ in model (1). In order not to have complete isolation of cases with HIV, one should take $\sigma_n > \frac{1}{\sqrt{2\beta\log(n)}} \approx 0.409$. For $\sigma_n = 0.5$, consider a repetition of the case of $n = 2,000$ and $\varepsilon = 1/2$, as well as the case of $n = 2,000$ and $\varepsilon = 0.05$. Thus $1/\sigma = 2$ and to control the FDR at 5%, one must evaluate $\frac{\epsilon}{1-\epsilon} \left(\frac{\bar{\Phi}(t-2)}{\bar{\Phi}(t)} \right) = 19$, which yields $t = 0.54$. Correspondingly, $n(1-\epsilon)\bar{\Phi}(t) = 2,000 \times \frac{1}{2} \times 0.029 \approx 29$, and $n\epsilon\bar{\Phi}(t-2) = 2,000 \times \frac{1}{2} \times 0.54 \approx 540$. Furthermore, for $k = 0,3,6,9$, the probability that no more than k “1”s are correctly classified are extremely small ($< 10^{-315}$). For the case of $n = 2,000$ and $\varepsilon = 0.05$. Similarly, one must similarly evaluate $\frac{\epsilon}{1-\epsilon} \left(\frac{\bar{\Phi}(t-2)}{\bar{\Phi}(t)} \right) = 19$, which yields $t = 3.62$. Correspondingly, $n(1-\epsilon)\bar{\Phi}(t) = 2,000 \times 0.95 \times 1.47 \times 10^{-4} \approx 0.28$, and $n\epsilon\bar{\Phi}(t-2) = 2,000 \times 0.05 \times 0.053 \approx 5$. Furthermore, for $k = 0,3,6,9$, the probability that no more than k “1”s are correctly classified are 4.5×10^{-3} , 0.22, 0.73, 0.96. Take $k = 3$. The probability that more than three “1”s are correctly classified is about 78%.

This example illustrates how the multiplicity issue arises when an intruder tries to match many records with those in a database “protected” by matrix masking. Simply protecting each individual record with high probability is not enough; there remains a substantial chance for one or more record to be vulnerable to disclosure.

Summary

The accumulation of massive data sets and the rapid development of the Internet expanded opportunities for data analyses as well as created enormous challenges for privacy protection. This brief overview of the literature on statistical disclosure limitation has stressed four

categories of approaches: sampling, perturbation, collapsing (aggregation), and the use of synthetic data. An overlooked issue in privacy protection is the notion of “multiplicity,” which is present whenever one attempts to protect many records simultaneously, or a data intruder tries to match the records of multiple targets in a database simultaneously. Simply protecting each individual record with high probability does not automatically protect all records, and careful statistical measures for resolving the multiplicity issue are necessary.

Key Applications

The methods of statistical disclosure limitation outlined here are already in widespread use by government statistical agencies throughout the world. The volumes by Doyle et al. [5] and Willenborg and de Waal [15] summarize a number of the approaches and methodologies. In particular, census data released by most developed countries are protected using these methods.

Future Directions

The elaboration of approaches described here to deal with very large scale databases remains a challenge, especially in the face of demands for increased access to data and novel attacks on databases by intruders. This entry presents the first known application of ideas and results from the multiplicity literature to the problem of statistical disclosure limitation and risk-utility tradeoff. These ideas require further development and integration with the rest of the literature. And methodology dealing with the risk-utility tradeoff will clearly need to evolve in response to the evolution of intruder strategies to compromise databases.

Cross-references

- ▶ [Data Perturbation](#)
- ▶ [Individually Identifiable Data](#)
- ▶ [Inference Control in Statistical Databases](#)
- ▶ [Matrix Masking](#)
- ▶ [Privacy](#)
- ▶ [Privacy-Preserving Data Mining](#)
- ▶ [Randomization Methods to Ensure Data Privacy](#)

Recommended Reading

1. Abramovich F, Benjamini Y, Donoho D, and Johnstone I. Adapting to unknown sparsity by controlling the false discovery rate. *Ann. Stat.*, 34:584–653, 2006.
2. Anderson M. and William Seltzer W. Challenges to the confidentiality of U.S. federal statistics, 1910–1965. *J. Off. Stat.*, 23:1–34, 2007.

3. Benjamini Y. and Hochberg Y. Controlling the false discovery rate: A practical and powerful approach to multiple testing. *J. Roy. Statist. Soc. B*, 57:289–300, 1995.
4. Dalenius T. Towards a methodology for statistical disclosure control. *Statistisk Tidskrift*, 5:429–444, 1977.
5. Donoho D. and Jin J. Higher Criticism for detecting sparse heterogeneous mixtures. *Ann. Stat.*, 32:962–994, 2004.
6. Doyle P., Lane J.I., Theeuwes J.J.M., and Zayatz L. (eds.). Confidentiality, Disclosure and Data Access: Theory and Practical Application for Statistical Agencies. Elsevier, New York, NY, USA, 2001.
7. Fienberg S.E. Confidentiality, privacy and disclosure limitation. In Encyclopedia of Social Measurement, Vol. 1. Academic Press, San Diego, CA, USA, 2005, pp. 463–469.
8. Fienberg S.E. and Makov U.E. Confidentiality, uniqueness and disclosure limitation for categorical data. *J. Off. Stat.*, 14:485–502, 1998.
9. Fienberg S.E., Makov U.E., and Sanil A.P. A Bayesian approach to data disclosure: Optimal intruder behavior for continuous data. *J. Off. Stat.*, 13:75–89, 1997.
10. Fienberg S.E., Makov U.E., and Steele R.J. Disclosure limitation using perturbation and related methods for categorical data (with discussion). *J. Off. Stat.*, 14:485–502, 1998.
11. Fienberg S.E. and Slavkovic A.B. Preserving the confidentiality of categorical statistical databases when releasing information for association rules. *Data Min. Knowl. Discov.*, 11:155–180, 2005.
12. Hertzog T.N., Scheuren F.J., and Winkler W.E. Data Quality and Record Linkage Techniques. Springer-Verlag, New York, NY, USA, 2007.
13. Lambert D. Measures of disclosure risk and harm. *J. Off. Stat.*, 9:313–331, 1993.
14. Raghunathan T.E., Reiter J., and Rubin D.B. Multiple imputation for statistical disclosure limitation. *J. Off. Stat.*, 19:1–16, 2003.
15. Warren S. and Brandeis L. The right to privacy. *Harvard Law Rev.*, 4:193–220, 1890.
16. Willenborg L. and de Waal T. Elements of Statistical Disclosure Control. Vol. 155. Lecture Notes in Statistics Springer-Verlag, New-York, NY, USA, 2001.

Steganography

RADU SION
Stony Brook University, Stony Brook, NY, USA

Synonyms

[Information hiding; Covert communication](#)

Definition

Steganography (from the greek “stegano” – covered) is a term denoting mechanisms for hiding information within a “cover” such that, generally, only an intended recipient will (i) have knowledge of its existence, and (ii) will be able to recover it from within its cover. In

modern digital steganography applications, the cover is often a multimedia object such as an image that is minorly altered in the steganographic process. Steganographic techniques have been deployed for millennia and several primitive war-time instances are described in the Histories of Herodotus of Halicarnassus, including a case of a message tattooed on the shaven head of a slave, which, when covered with grown hair acted as an effective “cover” when traversing enemy lines.

Key Points

Steganography versus Watermarking

A common trend of term misuse is associated with steganography. Specifically, many sources consider the term “watermarking” as equivalent. This is incorrect. There are fundamental differences, from both application perspectives and associated challenges. Steganography usually aims at enabling Alice and Bob to exchange messages in a manner as stealthy as possible, through a hostile medium where Malory could lurk. On the other hand, Digital Watermarking is deployed by a rights holder (Alice) as a court proof of rights over a Work, usually in the case when an adversary (Mallory) would benefit from using or selling that very same Work or maliciously modified versions of it. In Digital Watermarking, the actual value to be protected lies in the Works themselves, whereas pure steganography usually makes use of them as simple value “transporters.” In Watermarking, Rights Assessment is achieved by demonstrating (with the aid of a “secret” known only to Alice – “watermarking key”) that a particular Work exhibits a rare property (“hidden message” or “watermark”). For purposes of convincing the court, this property needs to be so rare that if one considers any other random Work “similar enough” to the one in question, this property is “very improbable” to apply (i.e., bound false-positives rate). It also has to be relevant, in that it somehow ties to Alice (e.g., by featuring the bit string “(c) by Alice”). There is a threshold determining the ability to convince the court, related to the “very improbable” assessment. This defines a main difference from steganography: from the court’s perspective, specifics of the property (e.g., watermark message) are not important as long as they link to Alice (e.g., by saying “(c) by Alice”) and, she can prove “convincingly” it is she who induced it to the (non-watermarked) original. In watermarking, the emphasis is on “detection” rather

than “extraction.” Extraction of a watermark, or bits of it, is usually a part of the detection process but just complements the process up to the extent of increasing the ability to convince in court.

Fingerprinting

In this application of steganography, license violators are “tracked” by hiding uniquely identifying “fingerprints.” If the Work would then be found in the public domain, the fingerprints can then be used to assess the source of the leak.

Recommended Reading

1. Watermarking World. Online at <http://www.watermarkingworld.org/>

Stemming

CHRIS D. PAICE

Lancaster University, Lancaster, UK

Synonyms

[Suffix stripping](#); [Suffixing](#); [Affix removal](#); [Word conflation](#)

Definition

Stemming is a process by which word endings or other affixes are removed or modified in order that word forms which differ in non-relevant ways may be merged and treated as equivalent. A computer program which performs such a transformation is referred to as a *stemmer* or *stemming algorithm*. The output of a stemming algorithm is known as a *stem*.

Historical Background

The need for stemming first arose in the field of information retrieval (IR), where queries containing search terms need to be matched against document surrogates containing index terms. With the development of computer-based systems for IR, the problem immediately arose that a small difference in form between a search term and an index term could result in a failure to retrieve some relevant documents. Thus, if a query used the term “explosion” and a document was indexed by the term “explosives,” there would be no match on this term (whether or not the document would actually be retrieved would depend on the logic and remaining terms of the query).

The first stemmer for the English language to be fully described in the literature was developed in the

late 1960s by Julie Beth Lovins [11]. This has now been largely superseded by the Porter stemmer [14], which is probably the most widely used, and the Paice/Husk stemmer [12]. Stemmers have also been developed for a wide variety of other languages.

Foundations

Definitions

In an IR context, the process of taking two distinct words, phrases or other expressions and treating them as semantically equivalent is referred to as *conflation*. The two expressions need not be precisely synonymous, but they must refer to the same core concept (compare “computed” and “computable”). In this article, the term “practically equivalent” is used to mean that, *for the purposes of a particular application*, the words may as well be taken as equivalent.

The term conflation is sometimes used as though it is equivalent to stemming, but it is in fact a much broader concept, since it includes (i) cases where the strings concerned are multi-word expressions, as in “access time” and “times for access”, and (ii) cases where the strings are not etymologically related, as in “index term” and “descriptor”. In case (i) special string matching techniques may be used, whereas in case (ii) reference to a dictionary or thesaurus is necessary. The present account deals exclusively with the conflation of etymologically related single words.

There are various possible approaches to word conflation, including the following.

1. *Direct matching*. In this method, the character sequences of two words are compared directly, and a similarity value is computed. The words are then considered to match if their mutual similarity exceeds a predefined threshold. To give a simple example, the first six letters of the words “exceeds” and “exceeded” are the same, so these words together contain 12 matching letters out of 15. Hence, a similarity of $12/15 = 0.80$ can be computed. Use of a threshold (say, 0.70) allows a decision as to whether the words can be considered equivalent.

With such a method, setting the threshold is problematic. Thus, the similarity between “exceeds” and “excess” is 0.62, which is below the stated threshold. However, allowing for this by lowering the threshold to 0.60 would cause “excess” and “except” (similarity 0.67) to be wrongly conflated.

2. *Lexical conflation.* In this case a thesaurus or dictionary is used to decide whether two words are equivalent. Obviously, this method can be used even for etymologically unrelated words. A problem here is obtaining a suitably comprehensive and up-to-date thesaurus, and one which explicitly lists routine variants such as plurals.
3. *Cluster-based conflation.* This method, investigated by Xu and Croft [15], involves creating clusters of practically equivalent words by analyzing the word-word associations in a large representative text corpus. Each query word is then supplemented by adding in the other words in its cluster. In contrast to method (2), the clusters created are specific to the text collection in question. However, the creation of the clusters can be very time-consuming.
4. *N-gram conflation.* In this method, each word is decomposed into a collection of N -letter fragments (N -grams), and a similarity is computed between the N -gram collections of two words; a threshold is then applied to decide whether the words are equivalent. This approach was pioneered by Adamson and Boreham [1], who used sets of *bigrams*, where $N = 2$. For example, after eliminating duplicates and sorting into order, “exceeds” can be represented by the bigram set {ce, ds, ed, ee, ex, xc} and “exceeded” by {ce, de, ed, ee, ex, xc}. Out of 7 distinct bigrams here, 5 are shared between the two words; hence a similarity of $5/7 = 0.712$ can be computed.
5. *Stemming.* Stemming refers to the removal of any suffixes (and sometimes other affixes) from an input word to produce a *stem*. Two words are then deemed to be equivalent if their stems are identical. This method is much favored because it is fast: all words can be reduced to stems on input to the system, and simple string matching used thereafter. The remainder of this article focuses on stemming in this narrow sense.

Stemming Algorithms

The most primitive type of stemming is *length truncation*, in which any word containing more than N letters is represented by its first N letters. Thus, using $N = 6$, “exceeds” and “exceeded” are both reduced to “exceed,” though “excess” remains distinct. Most stemmers, however, use rules which test for specific endings which, if found, are removed or replaced.

It is possible to implement a set of stemming rules by encoding them directly as a computer program.

This permits an arbitrary level of complexity in the tests and transformations used, but it makes the stemmer harder to design and modify. An alternative approach is to hold the rules in one or more tables, with a *stemming engine* designed to operate on those tables. This separation means that the same stemming engine can in principle be used with different rules on different occasions, depending on the particular stemming requirements. It also means that a given stemming engine can be used, with little or no adaptation, for a range of other languages.

In all of the stemmers to be described below, a stemming operation is subject to “acceptability” constraints. This ensures that if the ending of a word matches an ending in a table, the indicated action is only taken if relevant conditions are satisfied. These constraints vary from one stemmer to another (and sometimes from one ending to another). The Paice/Husk stemmer [12] uses a very simple constraint: an action only proceeds if the resulting stem will contain at least two letters, including at least one vowel. Thus, “string” cannot be transformed to “str” through a hit with an “-ing” rule.

It is important to note that, for the purposes of most applications, the stem returned by a stemmer need not be an actual word of the language. The essential desideratum is that the stem should be the same for all practically equivalent words, and different for all other words.

Stemming algorithms can be classified roughly as *single-stage*, *multi-stage* or *iterative*. Lovins’ stemmer [11] is often described as a single-stage stemmer, since it uses a single table containing all the distinct endings which are to be removed. In this table, the rules are held in decreasing order of length, and the first matching rule is the one applied. This ensures that, if the table contains the endings “-mentary,” “-ment” and “-ary,” the word “documentary” and “document” are both reduced to “docu.” If the “-ary” ending were tested first, “documentary” would simply be stemmed to “document.”

In fact, Lovins’ stemmer has a second (iterative) stage, using a table of 35 “recoding rules,” which can adjust the stem returned by the first stage, e.g., by replacing double final consonants by single, and making other changes – thus,

admission → admiss → admis
admittance → admitt → admit → admis

The 290 endings listed for the original Lovins’ stemmer are demonstrably inadequate, but a satisfactory rule set

would need to be much longer, and would show considerable structural redundancy. The Paice/Husk stemmer [12] avoids this by taking an iterative approach, where long endings are removed or transformed in a series of actions using a much shorter table containing shorter endings. When one action has been activated and completed, the table may be entered again to see if another rule will fire. Thus, some endings are removed in several stages:

$$\begin{aligned} \text{sensibilities} &\rightarrow \text{sensibility} \rightarrow \text{sensibil} \\ &\rightarrow \text{sensibl} \rightarrow \text{sens} \end{aligned}$$

The most popular stemmer for English is that devised by Martin Porter at Cambridge University [14]. This stemmer, which was designed to reflect the linguistic structure of suffixes in English words, proceeds through five main stages. Stage 1 deals with plurals, verb inflections, and words ending with “-y”; stages 2–4 with all the major derivational endings; and stage 5 with “-e” removal and singling of final “-ll.” Acceptability constraints are based on a quantity called the “measure” of the word, which is derived by scanning the consonant/vowel pattern of each word.

To facilitate the development of stemmers, Porter developed a special language known as Snowball (see URL below).

Krovetz developed a stemmer KSTEM which uses a machine readable dictionary to decide whether a stemmed form corresponds to an acceptable root form of the original word [9]. Despite careful refining of the algorithm to allow for a range of problem cases, IR performance was not consistently better than with Porter’s algorithm.

The design of a stemmer for a new language can be a labor-intensive business. An attractive alternative is to generate a set of stemming rules automatically. Thus, Bacchin et al. have developed a probabilistic approach which uses a large representative corpus to determine the optimal splitting of words into “stems” and “derivations.” They showed that, in terms of retrieval performance, their approach was about as good as Porter’s algorithm for a range of European languages [5].

Prefixes and Inflection

In English, stemmers are usually designed for removing suffixes from words. The removal of “intimate” prefixes such as “intro-,” “pro-” and “con-” generally results in words being wrongly conflated (consider

“intro-duction,” “pro-duction” and “con-duction”). However, there may be a case for removing looser prefixes such as “hyper-” or “macro-.” Also, prefix removal may be desirable in certain domains with highly artificial vocabularies, such as chemistry and medicine.

As explained below, there are some languages in which removal or replacement of prefixes, or even infixes, is in fact essential.

Performance and Evaluation

Since stemmers were originally developed to aid the operation of information retrieval systems, it was natural that they were first assessed in terms of their effect on retrieval performance, as well as on “dictionary compression” rates. Researchers were frustrated to find that the effects on retrieval performance for English-language material were small and often negative [10]. Removal of “-s” and other regular inflectional endings might be modestly helpful, but use of heavier stemming could easily result in a loss of performance [7]. Work by Krovetz and by Hull showed that most benefit is obtained in cases where the document or the query is short [8,9].

Stemmers are not used only in IR systems, but in a wide range of natural language applications. A less “IR-oriented” general approach to measuring performance is to consider the number of actual stemming errors committed by an algorithm, and this forms the basis of a method developed by Paice [13]. Notice first that stemming errors are of two kinds: *understemming*, in which a pair of practically equivalent words are not conflated, and *overstemming*, in which two semantically distinct words are wrongly conflated. It is easy to see that these two types of error trade off against one another.

Paice’s method makes use of a collection of distinct words (typically derived from an actual text source) which have been manually collected into groups, such that all the members of a group are practically equivalent. Two indices are computed based on a stemmer’s treatment of pairs of words, which reflect the rate of understemming and of overstemming. Moreover, the resulting values are related to a baseline represented by length truncation (see above). This results in a general measure of accuracy called the “error rate relative to truncation,” ERRT. Whilst this approach provides some insights into the activities of stemmers, it is unclear how such information should be used, though in future it might provide the basis for an optimization process. The use of human-defined target groups is a weak feature.

As a by-product, Paice's method yields a "stemming weight," which is the ratio of the overstemming to the understemming indices; a large stemming weight means that the stemmer is "heavy," "strong" or "aggressive." Frakes and Fox [6] present a series of other metrics related to stemming weight, as well as metrics for comparing stemmers one with another. These are all "behavior metrics," and do not relate directly to the actual accuracy of the stemming process.

Non-English Stemmers

Stemming is appropriate for most (though not all) natural languages, and appears to be especially beneficial for highly inflected languages [9]. There is neither space nor need to describe non-English stemmers here, except to note that some languages exhibit much greater structural complexity, and this warrants special approaches. Thus, a typical Arabic word consists of a root verb of three (or occasionally four or five) consonants (e.g., "k-t-b" for "to write"), into which various prefixes, infixes and suffixes are inserted to produce specific variant forms ("katabna": "we wrote" and "kitab": "book"). Some researchers have concentrated on extracting the correct root from a word [3], but Aljlayl and Frieder have demonstrated that better retrieval performance is obtained by using a simpler "light stemming" approach, in which only the most frequent suffixes and prefixes are removed [4]. Their results showed that extraction of roots causes unacceptable levels of overstemming.

Key Applications

As noted earlier, stemmers are routinely used in information retrieval systems to control vocabulary variability. They also find use in a variety of other natural language tasks, especially when it is required to aggregate mentions of a concept within a document or set of documents. For example, stemmers may be used in constructing lexical chains within a text. Stemming can also have a role to play in the standardization of data for input to a data warehouse.

Data Sets

Useful resources can be found on the two websites noted below.

URL to Code

Stemming algorithms and other resources may be obtained from the following websites:

<http://www.snowball.tartarus.org/>
<http://www.comp.lancs.ac.uk/computing/research/stemming/>.

Cross-references

► Lexical Analysis of Textual Data

Recommended Reading

1. Adamson G.W. and Boreham J. The use of an association measure based on character structure to identify semantically related pairs of words and document titles. *Inf. Process. Manage.*, 10(7/8):253–260, 1974.
2. Ahmad F., Yusoff M., and Sembok M.T. Experiments with a stemming algorithm for Malay words. *J. Am. Soc. Inf. Sci. Technol.*, 47(12):909–918, 1996.
3. Al-Sughaiyer I.A. and Al-Kharashi I.A. Arabic morphological analysis techniques: a comprehensive survey. *J. Am. Soc. Inf. Sci. Technol.*, 55(3):189–213, 2004.
4. Aljlayl M. and Frieder O. On arabic search: Improving the retrieval effectiveness via a light stemming approach. In Proc. Int. Conf. on Information and Knowledge Management, 2002, pp. 340–347.
5. Bacchin M., Ferro N., and Melluci M. A probabilistic model for stemmer generation. *Inf. Process. Manage.*, 41(1):121–137, 2005.
6. Frakes W.B. and Fox C.J. Strength and similarity of affix removal stemming algorithms. *SIGIR Forum*, 37(1):26–30, 2003 (Spring 2003).
7. Harman D. How effective is suffixing? *J. Am. Soc. Inf. Sci.*, 42(1):7–15, 1991.
8. Hull D. A Stemming algorithms: a case study for detailed evaluation. *J. Am. Soc. Inf. Sci.*, 47(1):70–84, 1996.
9. Krovetz R. Viewing morphology as an inference process. *Artificial Intelligence*, 118(1/2):277–294, 2000.
10. Lennon M., Pierce D.S., Tarry B.D., and Willett P. An evaluation of some conflation algorithms for information retrieval. *J. Inf. Sci.*, 3:177–183, 1981.
11. Lovins J.B. Development of a stemming algorithm. *Mech. Transl. Comput. Linguist.*, 11:22–31, 1968.
12. Paice C.D. Another stemmer. *SIGIR Forum*, 24(3):56–61, 1990.
13. Paice C.D. A method for the evaluation of stemming algorithms based on error counting. *J. Am. Soc. Inf. Sci.*, 47(8):632–649, 1996.
14. Porter M.F. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
15. Xu J. and Croft W.B. Corpus-based stemming using cooccurrence of word variants. *ACM Trans. Inf. Syst.*, 16(1):61–81, 1998.

Step

► Activity

Stewardship

► Digital Curation

Stop-&-go Operator

NIKOS HARDAVELLAS, IPPOKRATIS PANDIS
Carnegie Mellon University, Pittsburgh, PA, USA

Synonyms

[Non-pipelineable operator](#)

Definition

A Stop-&-Go operator, or non-pipelineable operator, is a relational operator which cannot produce any result tuples unless it has consumed all of its input. A typical Stop-&-Go operator is the Sort operator. The usage of Stop-&-Go operators in the query execution plan limits the degree of operator-level parallelism.

Key Points

Some relational operators need to consume their entire input before they are able to produce tuples. These operators are called Stop-&-Go or non-pipelineable operators. A typical example of a Stop-&-Go operator is the Sort operator. To sort a set of tuples, the entire input set needs to be consumed before the operator can output the tuples in sorted order. There are many Stop-&-Go operators, such as various flavors of Join and Aggregation. For example, Hash Join is a Stop-&-Go operator because the Probe phase cannot start unless the Build phase has finished. Similarly, Sort-Merge Join is a Stop-&-Go operator because the Merge phase cannot start unless the Sort phase has finished.

The Stop-&-Go operators stop the flow of tuples in a pipelined execution, so their usage limits the degree of operator-level parallelism. Thus, a query optimizer that aims to achieve high levels of operator-level parallelism may choose to replace Stop-&-Go operators with more expensive – but pipelineable – operators [1,2] in the query execution plan.

Cross-references

- ▶ [Hash Join](#)
- ▶ [Operator-Level Parallelism](#)
- ▶ [Pipelining](#)
- ▶ [Query Plan](#)
- ▶ [Sort-Merge Join](#)

Recommended Reading

1. Graefe G. Encapsulation of parallelism in the volcano query processing system. In Proc. of the ACM SIGMOD Conf. on Management of Data, 1990, pp. 102–111.

2. Johnson R., Hardavellas N., Pandis I., Mancheril N., Harizopoulos S., Sabirli K., Ailamaki A., and Falsafi B. To share or not to share? In 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 351–362.

Stoplists

EDIE RASMUSSEN

University of British Columbia, Vancouver, BC, Canada

Synonyms

[Negative dictionary](#); [Stopwords](#)

Definition

Stoplists are lists of words, commonly called stopwords, which are not indexed in an information retrieval system, and/or are not available for use as query terms. A stoplist can be created by sorting the terms in a document collection by frequency of occurrence, and designating some number of high frequency terms as stopwords, or alternately, by using one of the published lists of stopwords available. Stoplists may be generic or domain specific, and are of course language specific. When a stoplist is used for indexing, as a document is added to the system, each word in it is checked against the stoplist (for example through dictionary lookup or hashing), and those which match are eliminated from further processing. In some systems, stopwords are indexed, but the stoplist is used to eliminate the words from processing when they are used as query terms.

Key Points

Hans Peter Luhn, in pioneering work on automatic abstracting, put forward the idea that certain words are too common to provide a significant discrimination value, instead contributing noise to the calculations, and should be excluded from consideration [6]. In his description of the processing needed to create Keyword-in-Context (KWIC) indexes, he described a “dictionary of insignificant words” which was to be excluded from processing. In his view, these insignificant words would include “articles, conjunctions, prepositions, auxiliary verbs, certain adjectives, and words such as “report,” “analysis,” “theory” and the like” [7]. This idea was incorporated in the 1960s in commercial KWIC indexes introduced by Biological Abstracts (BASIC) and Chemical Abstracts (Chemical Titles). At

Biological Abstracts the number of excluded terms varied, but grew to 1,000 words, although analysis showed that 14 words were enough to prevent 80% of the entries, and the tradeoff between reduction in (printed) index size and cost of dictionary lookup became a factor as the length of the stopword list increased [2]. The use of the term “stopword” seems to come from this application, where designation of a word as a stopword stops the corresponding index entries from being printed [9]. As electronic databases became available for searching, database vendors created lists of stopwords which were not indexed or available for use in searching. The lists used by commercial systems were usually quite short; for example, in the Dialog system, the list consists of only nine words: An, And, By, For, From, Of, The, To, With [1]. Other lists which were published and used in IR research contain several hundred words; for an example see Fox [3].

There have been dual arguments put forward for the use of a stopword list, or stoplist, in building an index. The first relates to efficiencies in storage and processing. Common words follow a Bradford distribution and therefore a relatively small number of words account for a relatively large number of word occurrences. Data will vary somewhat from one corpus to another, but a typical analysis might show, for instance, that six words account for 20% of a corpus or 250–300 words for 50% [3]. Therefore, removing these words from the inverted index in a text retrieval system significantly decreases the size of an uncompressed index, though it adds to the processing time needed to create the index since a dictionary lookup or other technique is needed to identify words as stopwords when the text is processed. However, Witten et al. [10] suggest that the storage savings are most obvious in an uncompressed index, and are much less significant if an appropriate compressed representation is used. Processing queries containing stopwords can also be expensive, since their frequency of occurrence results in very long lists of postings. However efficiencies in query processing can be introduced, such as sorting postings lists by term weight, so that processing can be terminated when term weights are small, as is the case with stopwords [8]. Therefore, current techniques can address to a large extent the problems associated with processing very common words in both indexes and queries.

The second rationale for using a stoplist is the claim put forward by Luhn, that these words have very little

power for semantic resolution, and therefore may contribute noise rather than meaning for retrieval purposes. However, current term weighting techniques greatly reduce the contribution of common words in ranking functions, and there are many situations where an inability to use stopwords as query terms makes it difficult if not impossible to perform an effective search. There are classic examples of searches composed entirely of stopwords, such as “AT&T,” “To be or not to be,” or where a stopword is critical to the query, for example the “A” in “Vitamin A.” In other situations the removal of stopwords makes it impossible to adequately specify the query, for example, where common words are needed to clarify the relationship between terms. One approach, as used by Google for instance, is to index stopwords but to process them in queries only when the searcher specifically requests it in the query formulation, or when the query is composed only of stopwords [4]. This allows the stopwords to be used when they would be helpful, and ignores them when they are not, but it does require some knowledge of advanced search techniques on the part of the searcher. Overall, improved storage and compression techniques, term weighting schemes, and advanced query processing techniques significantly reduce the cost of including stopwords in a text retrieval system and arguments can be made for eliminating the stopword list [8,10].

Cross-references

- ▶ [Index Creation and File Structures](#)
- ▶ [Lexical Analysis of Textual Data](#)

Recommended Reading

1. Dialog Online Courses: Glossary of search terms. Available at: http://training.dialog.com/onlinecourses/glossary/glossary_life.html
2. Flood B.J. Historical note: the start of a stop list at Biological Abstracts. *J Am Soc. Inf Sci.*, 50(12):1066, 1999.
3. Fox C. /Lexical analysis and stoplists. In *Information Retrieval: Data Structures and Algorithms*, W.B. Frakes, R. Baeza-Yates, (eds.). Prentice-Hall, Englewood Cliffs, NJ, 1992, pp. 102–130.
4. Google Web Search Help Center. Search basics: use of common words. Available at: <http://www.google.com/support/bin/answer.py?answer=981>
5. Korfhage R.R. *Information Storage and Retrieval*. Wiley, NY, 1997.
6. Luhn H.P. The automatic creation of literature abstracts. *IBM J. Res. Development*, 2:157–165, 1958.
7. Luhn H.P. Keyword-in-context index for technical literature. *Am. Doc.*, 11(4):288–295, 1960.

8. Manning C.D., Raghavan P., and Schütze H. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.
9. Parkins P.V. Approaches to vocabulary management in permuted-title indexing of Biological Abstracts. In Proc. American Documentation Institute, 26th Annual Meeting, 1963, pp. 27–29.
10. Witten I.H., Moffat A., and Bell T.C. *Managing Gigabytes: Compressing and Indexing Documents and Images* (2nd ed.). Morgan Kaufmann, San Francisco, CA, 1999.

Stopwords

► Stoplists

Storage Access Models

KALADHAR VORUGANTI

Network Appliance, Sunnyvale, CA, USA

Synonyms

[Database storage layer](#); [Database physical layer](#)

Definition

Database management systems provide storage that can be accessed via a query language interface and that can be updated under the control of a transaction management system. A database management system can reside on top of a file system (system management storage) or on top of raw block storage (direct managed storage), or on a combination of file system and block storage (hybrid model). There are advantages and disadvantages of using these different types of underlying storage.

Historical Background

Database management systems have historically managed data on disks by themselves. Over the past few years, the management functionality in file systems has steadily improved. In order to leverage this management functionality (like data backup and recovery), and make it easier for system administrators to manage their storage infra-structure in a uniform manner, database system vendors started to architect database systems so that they can also run on top of file systems. This, in turn, has now provided users with multiple storage alternatives. These alternatives are discussed in this section.

Foundations

A database is logically organized into multiple table spaces. A table space determines the location from where a table gets its storage space. Thus, database tables are created inside table spaces. Multiple tables can exist in a table space. A table space obtains its storage from either files or a directory of a file system or multiple raw block devices. Each raw block device or file or directory that provides storage to a table space is known as a container. Multiple containers can provide storage to a single table space. Separate table spaces exist for user data, user index data, temporary data, and log data. Similarly, separate table spaces also exist for system tables and catalog tables. A table space can be classified as one of the following:

System Managed Space (file system): This type of storage container is fully under the control of the file system. The advantages of system managed space are:

- *In-time Provisioning:* Typically, database administrators use system managed storage for managing temporary storage. This ensures that space is only allocated when needed, and re-used for other purposes when it is de-allocated. This is an advantage of system managed space over database managed space, where space is pre-reserved in the containers.
- *Leverage File System Utilities:* System managed storage files can leverage backup, migration, and all of the other file system utilities. Previously, this was a major advantage for system managed storage, but over the past few years, a lot of progress has been made in building block level data management utilities that can be leveraged by a database managed system.

Database Managed Space (Raw Storage): This type of storage container is fully under the control of the database management system. The advantages of database managed space are:

- *Better Performance:* Database managed storage offers better performance because of the following reasons:
 - Unlike in system managed space, the absence of file system logging also helps with the overall performance.
 - As file systems age, the underlying managed storage system can become fragmented. The absence of file system fragmentation also helps with the performance of database managed storage.

- The lack of an intermediate file system buffer (in addition to the database buffer) in the I/O path, and the absence of contention in the file system buffer with other non-database applications, helps to improve the performance of database managed storage. With the recent emergence of direct I/O mechanisms, where the file system places data directly into the database buffer the disadvantages of an intermediate buffer have been reduced even in system managed storage space.
- Each database containers resource is dedicated to that container, and thus, there is no contention for storage space with other non-database applications.
- Storage Extensibility:* Database managed storage allows for dynamic addition of more storage containers to a table space. Thus, one does not have to *a priori* know the maximum size of the required storage space.
- Concurrent Access:* Some file systems put a limit on the number of concurrent accesses on a file (container). These limitations are not present in database managed containers.

Hybrid Space (database managed file): In this type of storage container, the file system created file is given to the database management system to manage. It is a compromise between the above two types of containers. In these containers, there is no intermediate file system buffer but the size of the container is limited to the size limits of the created file. The performance of hybrid containers is almost as good as the database managed containers, and one can leverage the conventional file system provided backup/migration utilities.

Key Applications

OLTP applications that are performance sensitive typically use system managed storage. Applications store their temporary storage in system managed containers.

Cross-references

- ▶ [Backup and Restore](#)
- ▶ [Buffer Management](#)
- ▶ [Database Concurrency](#)
- ▶ [Logging and Recovery](#)

Recommended Reading

1. Mellish B., Aschoff J., Cox B., and Seymour D. IBM ESS and IBM DB2 UDB Working Together. IBM Redbook, SG24-6262-00, San Jose, CA, 2001.

Storage Area Network

KAZUO GODA

The University of Tokyo, Tokyo, Japan

Synonyms

[SAN](#)

Definition

A Storage Area Network is a network whose main purpose is to transfer data between storage devices and servers and among storage devices. The term Storage Area Network can be a synonym of the term Storage Network, but differs in that the term Storage Area Network is usually identified with a network with block-level I/O services rather than file access services. More specifically, the term Storage Area Network is often used to refer to a network with Fiber Channel technology. However, SNIA released a more general definition in which the term Storage Area Network is not connected with any specific types of network connections. Under this definition, an Ethernet-based network infrastructure for mainly connecting storage devices could also be considered a Storage Area Network. The term Storage Area Network is often abbreviated to SAN. When the term SAN is used in connection with a specific network technology X, the use of a term "X SAN" is encouraged. A SAN based on Fiber Channel technology is sometimes referred to as Fiber Channel SAN. A SAN based on TCP/IP technology is often shortened to IP SAN. Despite the original meaning, the term SAN is sometimes identified with a storage system which is also implemented using a network.

Key Points

A SAN is a general network for connecting storage devices, but as a matter of fact, currently most SANs are implemented on top of Fiber Channel technology. A typical SAN is composed of Fiber Channel switches, storage devices such as disk arrays and tape libraries, and Fiber Channel host bus adapter (HBA) cards that are installed into servers. Alternative network technologies such as iSCSI, IFCP and FCIP are used mainly in entry-level storage systems or in wide-area network connections.

Cross-references

- ▶ [Direct Attached Storage](#)
- ▶ [Network Attached Storage](#)
- ▶ [Storage Network Architectures](#)

Recommended Reading

- Clark T. Designing Storage Area Networks: A Practical Reference for Implementing Fibre Channel and IP SANs. Addison-Wesley, Reading, MA, 2003.
- Storage Network Industry Association. The Dictionary of Storage Networking Terminology. Also available at: <http://www.snia.org/>.
- Troppens U., Erkens R., and Müller W. Storage Networks Explained. Wiley, New York, 2004.

Storage Array

- Redundant Array of Independent Disks (RAID)

Storage Broker

- Request Broker

Storage Consolidation

HIROSHI YOSHIDA

Fujitsu Limited, Yokohama, Japan

Definition

The processes of centralizing the storage infrastructure resources of multiple servers to reduce management costs, achieve better service levels, and strengthen control over data.

Key Points

In small scale IT systems, each server has its own dedicated storage infrastructure (internal disks or DAS). However, as server numbers and the amount and importance of business data stored in the storage infrastructure increases, managing such dedicated storage infrastructure resources per server becomes difficult and expensive. To solve this problem, the dedicated infrastructure resources of servers are centralized to storage infrastructure resources shared by all servers, using storage networking technologies such as SAN and/or NAS.

Once dedicated storage infrastructure resources are consolidated to SAN and/or NAS resources, storage

management operations can also be consolidated and centralized. For example, data backup is performed only once for the consolidated storage instead of individually for each server. This greatly reduces the cost of storage management. Another advantage is that expensive storage solutions such as disaster recovery using replicated data in remote sites can be shared by multiple servers with consolidated storage. The result is much improved data availability that can be achieved cost-effectively. In addition, from a data management and data security viewpoint; rather than having data spread over multiple servers, often managed by multiple administrators or divisions and based on disparate policies, data stored in consolidated storage can be managed based on more consistent policies.

Cross-references

- DAS
- ILM
- NAS
- SAN
- SRM
- Storage Network Architectures
- Storage Virtualization

Storage Controllers

- Storage Devices

Storage Devices

KALADHAR VORUGANTI

Network Appliance, Sunnyvale, CA, USA

Synonyms

Tapes; Tape libraries; CDs; DVDs; Optical storage; WORM; Storage controllers; NAS servers; Storage servers; Flash

Definition

One of the goals of database, file and block storage systems is to store data persistently. There are many

different types of persistent storage devices technologies such as disks, tapes, DVDs, and Flash. The focus of this write-up is on the design trade-offs, from a usability standpoint, between these different types of persistent storage devices and not on the component details of these different technologies.

Historical Background

From a historical standpoint, tapes were the first type of persistent storage followed by disks, CDs, DVDs, and Flash. Newer types of memory technologies such as PRAM and MRAM are still in their infant stages. These newer non-volatile memory technologies promise DRAM access speeds and packaging densities, but these technologies are still too expensive with respect to cost/gigabyte.

Foundations

- *Tapes/Tape Libraries:* Tape readers/tape head, tape library, tape robot, and tape cartridge are the key components of a tape subsystem. Tapes provide the best storage packaging density in comparison to other types of persistent storage devices. Tapes do not provide random access to storage. Data on tapes can be stored either in compressed or uncompressed format. Unlike disks, tape cartridges can be easily transported between sites. Most organizations typically migrate data from older tape cartridges to newer tape cartridges once every 5 years to prevent data loss due to material degradation. One can employ disk based caches in front of tape subsystems in order to allow for tapes to handle bursty traffic. Tapes that provide Write-Once, Read Many (WORM) characteristics are also available. WORM tapes are useful in data compliance environments where regulations warrant guarantees that a piece of data has not been altered. DLT and LTO are currently the two dominant tape technologies in the market. Technology wise both these standards have minor differences. Finally, from a pure media cost standpoint, tapes are less expensive (cost per gigabyte) than disks and other forms of persistent media.
- *Disks/Storage Controllers/NAS Boxes:* Disks are the most widely used form of persistent storage media. Disks are typically accessed by enterprise level applications when they are packaged as part of the processing server box (direct attached storage model), or are part of a network attached storage box (NAS) and accessed via NAS protocols or, are packaged as part of a storage controller box and accessed via storage area network protocols (SAN). The current trend is for protocol consolidation, where the same storage controller provides support for both SAN and NAS protocols. Typically, the size of the storage controllers can vary from a few terabytes to hundreds of terabytes (refrigerator sized storage controllers). A storage controller typically consists of redundant processors, protocol processing network cards, and RAID processing adapter cards. The disks are connected to each other via either arbitrated loop or switched networks. Storage controllers also contain multi-gigabyte volatile caches. Disks are also packaged as part of laptops. There is a marked difference in the manufacturing process, and testing process between the enterprise class disks and commodity laptop class disks. Disks vary in their form factor, rotational speed, storage capacity, number of available ports, and the protocols used to access them. Currently, serial SCSI, parallel SCSI, serial ATA and parallel ATA, Fiber Channel, and SSA are the different protocols in use for accessing disks. Lower RPM and disk idle mode are new disk spin-down modes that allow disks to consume less power when they are not actively being used.
- *DVD/Juke Boxes:* DVDs and CDs are optical storage media that provide random access and WORM capabilities. Only recently, the multiple erase capacity of an individual CD, or DVD was less than the capacity of a single disk drive or tape cartridge. DVDs can store more data than a CD, and a high definition DVD can store more data than a DVD. There are numerous competing standards for CDs, DVDs and high definition DVDs, however, format agnostic DVD players and DVD writers are emerging. Usage of DVDs is more prominent in the consumer space rather than in the enterprise space. A juke box system allows one to access a library of CDs or DVDs. DVDs have slower access speeds than most types of disks.
- *Flash/SSDs/Hybrid Disks:* Flash is memory technology that has non-volatile characteristics. Flash memory has slower read times than DRAM. Moreover, it has much slower write times than DRAM. One has to perform an erase operation before one

can re-use a flash memory location. One can only perform a limited number of erase operations. Thus, the number of write operations determines the Flash memory life. SLC and MLC are the two different NAND flash technologies. SLC can be erased a greater number of times, and it has faster access times than MLC based flash. NAND flash has faster write and erase times than NOR flash. NOR flash has faster read times than NAND flash. NAND flash is used to store large amounts of data whereas NOR flash is used to store executable code. People are using MLC flash in cameras and digital gadgets, and are using SLC flash as part of solid state disks (SSDs). SSDs provide block level access interface (SCSI), and they contain a controller that performs flash wear leveling and block allocation. Hybrid disks that contain a combination of disks and Flash are emerging. Hybrid disks provide a Flash cache in front of the disk media. One typically can store meta-data or recently used data in the flash portion of hybrid disks to save on power consumption. That is, one does not have to spin-up the disk. Flash storage provide much better random access speeds than disk based storage.

Key Applications

Tapes are being used primarily for archival purposes because they provide good sequential read/write times. Disks are the media of choice for most on-line applications. Optical media (CDs, DVDs) are popular in the consumer electronic space. Flash based SSDs are popular for those workloads that exhibit random IOs. Disks are being used in Laptops, desktops and storage servers (SANs, NAS, DAS). Tape based WORM media and content addressable based disk storage are providing WORM media capabilities in tape and disk technologies, and thus, these technologies can be used to also store compliance/regulatory data.

Cross-references

- ▶ [Backup and Restore](#)
- ▶ [Direct Attached Storage](#)
- ▶ [Network Attached Storage](#)
- ▶ [Storage Area Network](#)

Recommended Reading

1. Anderson D., Dykes J., and Riedel E. More than an interface-SCSI versus ATA. In Proc. 2nd USENIX Conf. on File and Storage Technologies, 2003.

2. Toigo J. *Holy Grail of Network Storage Management*. Prentice Hall, Englewood Cliffs, NJ, 2003.
3. Voruganti K., Menon J., and Gopisetty S. Land below a DBMS. ACM SIGMOD Rec., 33(1):64–70, 2004.

Storage Grid

KALADHAR VORUGANTI

Network Appliance, Sunnyvale, CA, USA

Synonyms

[Data grids](#); [Content delivery networks](#); [Peer to Peer network](#); [Distributed databases](#); [Cloud computing](#); [Utility computing](#)

Definition

In grid computing, storage and computing resources are geographically spread out and accessed via fast wide-area networks. Storage resources could either co-exist with computing resources, or they could exist separately from the computing resources. Databases, file systems or block storage devices can be accessed remotely across fast wide-area network based grids. A storage grid provides services for discovering storage resources, transferring data, recovering from unfinished data transfer failures, data authentication/encryption services, and data replication services for performance and availability purposes. Storage grids also typically provide the necessary mapping layers to access data from heterogeneous sources. Heterogeneity can be due to differences underlying system architectures, data formats, protocols used to access the data, and data organization. Finally, storage grids provide a global unified namespace across the resources, and one typically transfers large datasets (multi-terabytes) across the different nodes.

Key Points

Content Delivery Networks (CDNs) is a related area, where data are cached at secondary servers at various geographically distributed servers to cut down on data access latency. Peer-to-Peer networks is another related area where data are distributed across different peers in an ad-hoc manner and there is no central authority. Storage grids can be further classified based on the following criteria [1]:

Storage Grid Organization: Storage grid organization deals with how resources are organized in the system. Resources can be organized in a hierarchical

manner, or in a monadic manner. In hierarchical model data exists at multiple sites. Each site in turn decides which of its children sites can have access to data. In a monadic grid data exists only at a single location and everyone accesses data from that location. Data sources in a grid can also be arranged in a federated model, where each site retains independent control over the data and its participation in the grid.

Data Transport: Data transport deals with transport issues, security issues, and fault-tolerance issues:

- *Transport:* Data can be transferred using protocols such as FTP and GridFTP. In addition, one could potentially employ overlay networks that provide caching functionality and control to the applications to directly control data transfer. Data can also be transferred via multiple parallel streams from one source location, and in a striped manner from multiple data source locations.
- *Security:* Authentication, encryption and authorization are the three security issues that are also applicable in grid environments.
- *Fault-Tolerance:* The primary fault-tolerance approaches are to restart a failed data transfer, or to have the ability to resume from the failed point. In some environments, if the destination node is not available, intermediate caches can temporarily store the data and then forward the data once the unavailable node comes back on line.

Data Replication and Storage: Data replication strategies can be classified in the following different ways:

- *Method:* Synchronous and asynchronous replication strategies are the two major classes of data replication mechanisms. In synchronous replication, updates are not acknowledged at the source until data has been successfully copied at the target locations. In asynchronous protocols, updates are immediately acknowledged at the source, and there is a lag in data consistency between the primary and secondary data copies.
- *Protocols:* Some grids employ open data transfer replication protocols, such as FTP or GridFTP. In open protocols, the catalog management becomes the responsibility of the application. Others employ a closed protocol which perform catalog management in an integrated manner.
- *Replication Granularity:* Data can be replicated at dataset, file, block, and database table level.

- *Replication Strategy:* Data can be replicated dynamically based on an objective function such as response time requirements, load balancing requirements or data consistency requirements, or data can be replicated statically based on an a priori schedule or on demand.

Resource Allocation and Scheduling: The goal of resource allocation and scheduling is to ensure that the data are located at the appropriate site in order to meet the performance and availability goals of the application. The settings for the following parameters can be varied in this regard:

- *Process Model:* The processes can be scheduled as independent tasks, as a bag of tasks or as part of a workflow. Workflow corresponds to a sequence of tasks, whereas, a bag of tasks correspond to executing the same task on different input parameters.
- *Objective Function:* Resource allocation and task scheduling is performed based on an objective function. The objective function tries to optimize load balancing, or business profit, or application performance. The object function is assigned at the task, bag of tasks or workflow level.
- *Scope of the Scheduler:* The scheduler decides whether to replicate/migrate data/process can try to optimize the utility function at the level of an individual application or at a more global community level.
- *Types of Tasks:* Data can be migrated, replicated, cached or remotely accessed in order to satisfy application's storage requirements. Alternately, the computation process can be migrated to the location where the data exists.

Storage grids are primarily used in scientific computing environments that deal with large amounts of data. It is usually not practical to replicate all of the data, and thus, grid architecture facilitates the remote access of large datasets across wide-area networks. However, variants of storage grid architectures such as CDNs are used to transfer streaming video, and P2P networks are used to shared audio and video data. Cloud computing is the new variant of utility computing where application, storage and server resources are managed by a service provider and clients remotely access these resources.

Cross-references

- [Grid and Workflows](#)
- [Grid File](#)

- ▶ Resource Scheduling
- ▶ Storage Protocols
- ▶ Storage Security

Recommended Reading

1. Venugopal S., Buyya R., and Ramamohan Rao K. A Taxonomy of Data Grids for Distributed Data Sharing, Management, and Processing. *ACM Comput. Surv.*, 38(1):1–53, 2006.

Storage Layer

- ▶ Storage Manager

Storage Management

HIROSHI YOSHIDA

Fujitsu Limited, Yokohama, Japan

Definition

The methods and tools used to manage storage devices (disk arrays, tape libraries, etc.), storage networking devices (fiber channel switches, etc.), storage-related components inside servers (host bus adaptors, etc.), and logical objects mapped on those devices (logical units, access paths, etc.). In general, the scope of storage management is limited to the management of storage infrastructure and does not handle the data stored in the infrastructure. The functions of storage management include device management, performance management, and problem management. Those functions are usually provided as software tools.

Historical Background

Storage management technologies have developed in parallel with the evolution of storage networking.

In the early 1990s, storage devices were used as DAS devices. Even in a DAS environment, storage management functions such as storage device management were required, and those functions were provided as dedicated software tools for specific vendors and/or devices, and those tools were often bundled with the hardware.

With the evolution of storage area networks (SANs) in the late 1990s, new requirements for storage management arose:

- Initially, SAN brought significant reduction of management cost through its storage consolidation capability. The storage capacity which could be managed by storage administrators was also greatly increased, compared to DAS environments. However, along with the growth of SAN, the number of connected devices and the amount and business importance of stored data increased; causing SANs to become more and more complex. This increased the “storage management gap,” i.e., the gap between the decrease in storage hardware costs and the extreme increase in storage management costs. A reduction in the manpower required for storage management was urgently needed.
- To manage a SAN environment, both storage device management functions and network management functions, such as discovery, network configuration management, and network topology management, are necessary. In a networked environment, coping with hardware and software failures and performance problems also becomes much more difficult, compared to a DAS environment.
- SAN environments are usually constructed using products from multiple vendors. Storage management tools must cope with such multi-vendor environments. Therefore a new business model of providing software products which are independent from specific storage vendors’ devices was established. This requirement also accelerated the standardization of the interfaces between storage devices and management software.

Foundations

General Classification of Storage Management Functions

Although storage management is necessary for all types of storage networking, the management of SANs is mainly discussed in the following description. This is because SANs are the most commonly used storage infrastructures, and feature the most typical management requirements. In general SAN environments, storage management is achieved by software tools running on a management server. Those tools provide

functions such as device management, configuration management, performance management, and problem management.

- Device management and configuration management consists of functions to configure and to monitor storage devices (disk arrays, tape libraries, etc.), network devices (fiber channel switches, etc.), server components (fiber channel host bus adapters (HBAs), etc.), and relationship between those components. For example, disk array management provides the following functions:
 - Monitoring and displaying the status of devices
 - Creation/configuration/masking/mapping of logical units/logical unit numbers (LUNs)

Management of fiber channel switches and SAN configuration provides the following functions:

- Monitoring and displaying switch ports
- Collecting and displaying statistic information on switch ports
- Displaying SAN topology
- Configuration of zones
- Integrated and consistent control of multiple switches

As a SAN environment includes multiple levels of virtualization (e.g., disk arrays, volume managers, and virtualization network appliances) and access control features (zone, host affinity, and LUN mapping), the mapping of logical access paths on physical paths and the correlation between applications and physical storage devices tend to be complex. Configuration management should provide functions to visualize such mapping and correlation from the viewpoint of application and to configure multiple SAN components in a consistent manner.

- Performance management consists of functions to monitor, to analyze, and to display storage access performance based on statistic information collected from storage devices and fiber channel switches. It also includes functions to issue an alert in case that a specific parameter (e.g., device busy rate) exceeds the predefined threshold.
- Problem management consists of functions to monitor the status of storage devices as well as fiber channel switches and to notify the administrators

and/or remote maintenance centers when problems such as hardware failures are detected.

“Storage management” is a very generic term and those elemental storage management functions are sometimes named “storage x management,” e.g., “storage device management” and “storage configuration management.” Note that the term “Storage resource management” represents a different concept from storage management mentioned here. It is usually used to indicate the functions to visualize and control usage of storage systems from the more application-aware or content-aware management viewpoint. It is described as a separate article.

Actual storage management software products include those typical functions as well as additional functions such as automation and the provision of integrated monitoring and operational views.

Elemental Technologies of Storage Management

Another aspect of understanding storage management is the technologies needed to implement storage management software. In general, the following internal functions are commonly required to implement storage management software.

- *Discovery* is a function to find storage devices in an SAN environment before knowing the topology of the SAN. When a new device such as a disk array system is connected to an SAN, it also has to be discovered.
- *Data collection* acquires necessary information, once storage devices are discovered. Information is collected through proprietary interfaces and/or standard interfaces such as SNMP (Simple Network Management Protocol) and SMI-S (Storage Management Initiative Specification). Data collection also stores that information in storage management repositories which are usually located in the storage management server. Collected information includes both information on the current status of storage resources such as configuration information and historical information such as accumulated performance information on storage devices.
- *Topology management* analyzes the topology of SANs based on the collected information. Topology management handles both physical network topology which represents the relationship between physical resources such as HBA ports, switch

ports, and storage device ports and logical network topology which represents the relationship between logical and/or virtualized resources such as LUNs, zones, and logical access paths.

- *Visualization* is a set of functions which display the information mentioned above on a management console and provide human interfaces to enable administrators to monitor and configure the storage resources easily.
- *Event processing* receives asynchronous events from storage resources, categorizes them based on the predefined policies, and notifies administrators and/or remote maintenance centers if necessary, and records those events into event log files.
- *Security management* is a set of functions which are necessary to meet the appropriate security requirements. Security management includes authorization and access control features of administrators considering multiple administrative roles, single sign-on features among multiple storage management software products, management of credentials of managed storage resources for data collection, and logging functions for auditing all operations applied to the storage infrastructure to fulfill compliance requirements.

One important topic related to storage management implementation is how the interoperability between management software products and managed storage resources is achieved. At an early stage, storage management was implemented as management tools provided by individual storage vendors. Those tools were dedicated to the storage devices of respective vendors. Each storage vendor developed a proprietary interface and protocol which was applicable only to its storage devices and storage management software tools.

However, it became very common for datacenters to use storage devices provided by multiple vendors. Using multiple storage management tools with different looks-and-feels and manageability increased the management and labor costs in datacenters. This situation led to the new requirement that storage management software must be able to manage not only single vendor storage resources but multi-vendor storage resources. The goal was that every storage management software product could manage every storage resource. Standard interfaces and protocols which could be applied to the communication between management

software and storage resources was crucial to achieving this goal.

Since the storage industry became aware early on of the importance of interoperability between products, standard interfaces and protocols for storage management were established as ISO standard SMI-S by the storage standards body SNIA (Storage Networking Industry Association).

Key Applications

Storage management is one of the essential features for administrative practice of computer system storage infrastructure. It allows administrators to configure, monitor, and control storage resources, particularly in SAN environments.

Storage management is also necessary as the basis of implementing higher level management, which is described as “management applications” later. For example, datacenter management requires storage management as part of its resource management as well as server management and network management. Another example is that information management requires storage management to manage the infrastructure in which information resides. In information lifecycle management, optimal storage devices are assigned to store information in accordance with its business value. Storage management is responsible for establishing the multiple storage device tiers which meet the different service level objectives and cost requirements.

Future Directions

The following areas will become more important in terms of storage management.

Integration of Management Software Including Storage Management

Currently a huge variety of management software products are used in large datacenters. Storage management software products must be integrated with those products.

To achieve consistent and automated resource management in a datacenter, storage management should be integrated with other resource management such as server and network management. This integration of resource management will provide capability such as the “provisioning” of a set of servers, storage devices, and network resources to an application. When administrators use this feature, they will not need to have

detailed knowledge and skills in storage management, letting them manage storage devices in an SAN environment without regard to the SAN. This feature will be achieved through the cooperation of server management, network management, and storage management, which will configure internal connections between servers and storage devices automatically. Such integrated provisioning will greatly reduce management and labor costs in the datacenter.

Another example is integration with IT service management such as incident management, problem management, change management, release management, and configuration management. IT service management controls those management processes in a manner which is compliant to ITIL (Information Technology Infrastructure Library). In addition, configuration information needed for those management processes is stored in a CMDB (Configuration Management Database). To achieve integration with IT service management, storage management operations will have to be initiated as part of management processes by workflow managers, and configuration information on storage resources collected by storage management will also have to be federated with CMDB.

Visualization and Optimization from a Business Viewpoint

The business value proposition brought by IT systems becomes more and more important and needs to be clearly stated. On the other hand, to achieve business risk management, the impact of IT infrastructure outages on the customer's business also has to be clearly analyzed. To fulfill such requirements, the relationship between IT resources and business applications must be visualized. The IT resources must also be optimally configured and managed to achieve the performance and availability objectives of business applications. The most important key technology is dependency mapping between business applications and resources. Another important technology is policy management which allows customers to specify the criteria of system behavior based on their business requirements. Storage management is responsible for that capability with regard to storage resources.

Establishing Framework for Management Applications

In addition to the necessity of basic management capability such as the configuring and monitoring of storage devices, the importance of higher level "management

applications" is continuously increasing. For example, such management applications include:

- Database performance management
- Resource provisioning
- Lifecycle management of storage resources and information lifecycle management
- Security management
- Automated management such as run book automation and autonomic management such as self configuration

Management applications are essential for achieving integrated management and business-aware management. The amount of management applications which are available on a storage platform is a key factor in achieving strategic use of storage and stored data in an enterprise.

In general, management applications monitor and control storage resources on the level of logical resources not on physical storage device level. In the main the information directly collected with current storage management tools, from storage hardware and the control functions of storage hardware, are sometimes too detailed for management applications. For example, a management application which provides provisioning capability is only aware of LUNs, their capacities, and the zoning configurations which restrict accessibility between servers and storage resources. There are also functions which many management applications use commonly to implement their storage-related capabilities. High-level interfaces which allow management applications to handle storage resources more logically, as well as a set of common components which help the development of management applications, are strongly required.

The storage industry has started to provide the higher-level interfaces and functional components as frameworks for management applications. One example is SNIA's attempt to standardize "management framework" as a higher level interface than the current SMI-S. It also includes common functional components such as discovery, data collection, topology management, data models, and policy management. Another example is "Aperi," which is the open source storage management software provided by open source community Eclipse. Aperi also provides the higher level management functions which are implemented on the SMI-S based storage management to management applications.

URL to CODE

The latest documents on SMI-S can be downloaded from the SNIA web site.

http://www.snia.org/tech_activities/standards/curr_standards/smi/

An open-source framework of storage management software can be downloaded from the Aperi project of Eclipse.

<http://www.eclipse.org/aperi/>

Cross-references

- ▶ DAS
- ▶ ILM
- ▶ LUN
- ▶ Mapping
- ▶ Masking
- ▶ SAN
- ▶ SMI-S
- ▶ SRM
- ▶ Storage Consolidation
- ▶ Storage Network Architectures
- ▶ Storage Networking Industry Association
- ▶ Storage Protocols
- ▶ Storage Virtualization
- ▶ Volume
- ▶ Zoning

Recommended Reading

1. Cummings, R. Storage Network Management, Storage Network Industry Association, 2004. Available at: http://www.snia.org/education/storage_networking_primer/stor_mngmnt/
2. Storage Network Industry Association. Storage Network Industry Association tutorials, 2007. Available at: <http://www.snia.org/education/tutorials/>

Storage Management Initiative-Specification

HIROSHI YOSHIDA

Fujitsu Limited, Yokohama, Japan

Synonyms

SMI-S

Definition

A standard storage management interface developed by the Storage Networking Industry Association (SNIA). SNIA describes SMI-S as follows: SMI-S defines a

method for the interoperable management of a heterogeneous SAN, and describes the information available to a Web-Based Enterprise Management (WBEM) client from an SMI-S compliant Common Information Model (CIM) server and an object-oriented, XML-based, messaging-based interface designed to support the specific requirements of managing devices in and through SANs.

Key Points

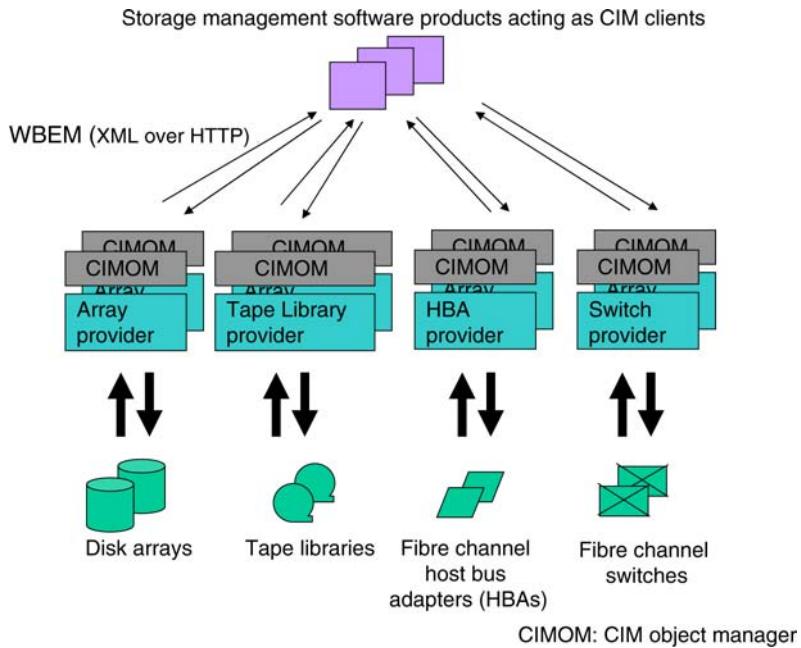
To implement storage management, methods to retrieve configuration information on storage components such as storage devices (disk arrays, tape libraries, etc.,), network devices (fiber channel switches, etc.,), and servers (hardware such as host bus adapters and software such as drivers and volume managers), plus methods to operate those components are necessary. To manage SAN environments composed of multi-vendor products, common methods of collecting information and operations must be standardized. In the standardization of those common methods, not only the standardized interfaces and protocols but also the semantics of the information and operations, i.e., object models of managed resources, must be defined.

To achieve SAN management in multi-vendor environments, the SNIA standardized on the Storage Management Initiative-Specification (SMI-S). Based on the Common Information Model (CIM) standardized by the Distributed Management Task Force (DMTF), SMI-S defines a common object model for each storage resource class as well as common methods of collecting information and operations using a Web-browser based management framework (Web-Based Enterprise Management, WBEM).

Figure 1 shows the architecture of SMI-S. It is based on the client-server model. Managed storage components act as CIM servers and storage management software tools act as CIM clients. CIM servers and CIM clients communicate by passing XML texts through HTTP. A SMI-S CIM server includes an HTTP server, an XML parser, an SLP agent used for discovery, and a CIM provider. The CIM provider implements actual CIM operations according to the profile defined for the corresponding storage resource class.

SMI-S version 1.3 includes the following storage management features and functionality:

Hardware Devices: SMI-S Providers
Switches



Storage Management Initiative-Specification. Figure 1. The architecture of SMI-S.

- Arrays (fiber channel and iSCSI)
- Servers
- NAS devices
- Tape libraries
- Host profiles
- Host bus adapters
- SMI-S Clients (software)
 - Configuration discovery
 - Provisioning and trending
 - Security
 - Asset management
 - Compliance and cost management
 - Event management
 - Data protection

The standardization effort began in 1997 and was adopted formally by the SNIA as SMI-S in 2002. It was designated as a standard by ISO/IEC in January 2007.

Cross-references

- [Storage Management](#)
- [Storage Networking Industry Association](#)

Recommended Reading

1. Storage Network Industry Association. Storage management initiative-specification, 2007. Available at: http://www.snia.org/tech_activities/standards/curr_standards/smi/

Storage Manager

GOETZ GRAEFE

Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

[Storage layer](#); [Disk process](#)

Definition

The storage manager is a software layer within a database management system. It relies on operating system primitives for I/O, synchronization, etc. and exposes records in storage structures such as B-trees and heaps. Its principal operations are creation and removal of storage structures as well as retrieval, search, scans, insertion, update, and deletion of records. For those operations, the storage layer provides concurrency control among threads and among transactions, as well as recovery from transaction, media, and system failures. Standard implementation techniques include locking and write-ahead logging. Using a buffer pool, records are made accessible in random access memory, although the permanent storage is on block-access media, typically on disk but possibly in flash memory.

The storage layer may support data compression, “blobs” (binary large objects), bitmaps in non-unique secondary indexes, hash indexes, multi-dimensional

indexes, etc. It may also manage the catalogs needed to manage its objects, or it may leave metadata management to a higher software layer. For all the supported data structures, the storage layer provides utilities such as backup and restore, bulk insertions and deletions, logical and physical consistency check, defragmentation and other forms of reorganization, etc.

Historical Background

Design and scope of the storage layer are often similar to those of the RSS (research storage system) of the System R prototype. The basic architecture of the core functions has remained unchanged, including space management for records in pages, indexing and search, concurrency control and recovery. Many specific implementation techniques have changed, e.g., introduction of multi-dimensional indexes. Another change has been the transition from write-ahead logging with recovery from a read-only log to recovery by compensation of logical actions and guaranteed exactly-once execution of physical actions, both logged and even checkpointed during database recovery.

The architecture of utilities has also undergone some changes. The implementation of many storage layer functions now employs query execution and even query optimization, e.g., memory management policies or partitioning and pipelining for parallel execution. For example, creation of a new secondary index may scan existing secondary indexes rather than the primary index, or consistency check may aggregate facts gathered during a disk-order scan rather than navigate each index with many random I/O operations.

Sorting used to be a storage layer function because it was used almost exclusively for index creation, traditionally a storage layer function. Sorting can be used for many tasks, however, including complex query execution plans with merge join operations, etc., such that sort- and hash-based operations should both be part of the query processing layer.

The plethora of features and functions has led to complexity and high total cost of ownership for many installations. Extensible database management systems have not solved this problem, and the tension between “one size fits all” and “tailor-made” database systems might be resolved through factoring and mass-customization.

XML support is now available in the storage layer of many commercial database management systems, but further improvements in query processing over

XML documents and databases will likely require further improvements of storage layer techniques.

In other words, as much as the storage layer may seem like a well-understood component of database technology, research, development, and competition continue unabated.

Foundations

This section describes a database system’s storage layer by its external interfaces above and below, followed by internal components and data structures on disk and in memory, and completed by specific techniques for query processing, concurrency control and recovery, utilities, and catalogs.

Storage Layer Concepts

The most important services provided by a database system’s storage layer revolve around indexes and records. Both of these are physical database concepts, quite different from logical database concepts such as table and row. For example, a single table may require many indexes due to redundant, secondary indexes as well as horizontal and vertical partitioning. Similarly, a single row might be represented by many records. Conversely, multiple tables may be clustered within a single index such that related information, e.g., about a purchase order and its line items, can be read, written, and buffered within a single disk page. Finally, a single record may contain information about multiple rows, e.g., a B-tree entry in a non-unique non-clustered index with a key value and a list of row identifiers.

The mapping between logical and physical concepts is usually provided by the relational layer and its query optimization component. Some implementations of the storage layer, however, provide some of this mapping, e.g., maintenance of all indexes for a row update, in order to maximize performance by minimizing the number of storage layer invocations.

Logical concepts include table, view, row, column, and domain; physical concepts include index, partition, record, and field. The word “key” is used both in the relational layer, where it restricts duplicates and null values, and in the storage layer, where it indicates index organization, sort order, or a search argument. It might be useful in some discussions to avoid the term “key” and to use “primary key,” “foreign key,” “index key,” and “search key” instead.

Storage Layer Services

The storage layer provides access and updates for indexes and their records. Heaps can be thought of as indexes that map a record identifier to a record, with no capability to modify the record identifier. Indexes can be created and dropped. Records can be inserted, deleted, updated, scanned, and searched using an appropriate search key. Some indexes, e.g., heaps and clustered indexes, may generate unique row identifiers during insertion and possibly a new one during update. Such row identifiers can link all records representing a logical row, e.g., during retrieval using a non-clustered index or during deletion of a logical row.

All operations are part of transactions with concurrency control and recovery. The storage layer supports pre-commit for participation in coordinated commits in addition to the traditional immediate commit. The set of currently active transactions and their current state may be managed by the storage layer or by another component within the database management system.

If multiple threads invoke the storage layer at the same time, internal data structures are protected against concurrency problems using appropriate low-level synchronization.

If catalogs are managed by a higher software layer, the storage layer provides appropriate indexes and, most likely, special high-performance lock scopes and lock modes for metadata. For example, all metadata about a table may be covered by a single lock, and this lock may cover both the metadata and the data. Specifically, the weakest mode merely read-protects the metadata, whereas the strongest mode permits arbitrary changes to both metadata and data. Traditional read and write locks (shared and exclusive) imply read-protection on the metadata.

Storage Layer Requirements

A storage layer invokes two basic functions of the operating system: input/output and synchronization. Asynchronous I/O functions are valuable as otherwise they must be simulated with additional threads. Synchronization primitives that provide both shared and exclusive levels directly support the need of database management systems and their storage layers.

Storage Layer Components

The well-known components of a storage layer are the access methods (B-tree structure etc.), buffer pool, concurrency control and recovery. The less prominent

components are memory management, disk space allocation, and catalogs. Asynchronous I/O could be separated into its own component, as could be latches, temporary structures (for sort- and hash-based operations), page structures (managing variable-length records), utilities for reorganization and consistency checks, backup and restore. Initial access to a database file, e.g., during recovery from a crash or when opening a database from a less-than-fully-trusted source, requires substantial logic and could be its own component, too.

Typically, multiple threads may invoke the storage layer at the same time; access and update of shared data structures within the storage layer is coordinated using latches. Those are equivalent to critical sections or locks in programming languages. Latch modes are typically shared and exclusive (read and write). Deadlock avoidance is required as latches do not participate in deadlock detection. Ordering latches based on levels is a standard technique to avoid deadlocks. Latches must not be retained while waiting for I/O to complete or while waiting for a database lock.

On-Disk Data Structures

The essential on-disk data structures are index for user data. This includes both primary, clustered indexes and secondary, non-clustered indexes on both tables and views. Additional data structures enable the essential ones: catalogs, free space management, and the recovery log. Each of those is special in their own way. For example, the recovery log is often mirrored on two devices in order to approximate the fiction of guaranteed stable storage.

Catalogs and their indexes often use the same on-disk format as user data. This is also possible for free space management, in particular if bitmap indexes are supported. The free space information is crucial during initial access to a database, i.e., system boot and recovery must be supported.

There is also the issue of free space management within each page. Each page typically consists of a page header, space for variable-length records, and an indirection vector that indicates the location of each record. A format version number in the page header enables incremental improvement of the database format without unloading and reloading entire databases. A “page LSN” (log sequence number) indicates the most recent log record pertaining to the page and is essential for exactly-once change application in modern recovery schemes.

Many database management systems and their storage layer support multiple index formats, e.g., hash indexes, heaps, multi-dimensional indexes, column storage, etc. All of them can be mapped to B-trees with reasonable efficiency in time and space, with a great savings in implementation effort. For example, implementation and testing for high scalability through a fine granularity of locking is a very substantial effort required for each storage structure.

In addition, bitmap indexes can be realized as a form of compression for non-unique non-clustered indexes. Even master-detail clustering (e.g., purchase orders and their line items) can be implemented relying merely on the sort order of B-trees and appropriate record formats. Finally, B-trees can be adapted to support “blobs” (binary large objects) for unstructured data. XML documents and other semi-structured data can be mapped to blobs plus traditional indexes for efficient search.

In-Memory Data Structures

The most prominent in-memory data structures within the storage layer are the lock manager’s hash table and the buffer pool including buffers for the recovery log. Other data structures enable transaction management, checkpoints, device management, and asynchronous I/O.

In addition to the direct images of on-disk pages in the buffer pool, a storage layer may cache high-traffic data in data structures designed for fast in-memory access. Catalogs and bitmaps for free space management are obvious candidates. In order to achieve “in-memory performance” for user data and their indexes, interior B-tree nodes can be augmented in the buffer pool with in-memory pointers to child nodes also in the buffer pool, in a special form of pointer swizzling. In all cases, update propagation between cache and disk page images in the buffer pool must be ensured.

Query and Update Processing

The storage layer serves query and update processing but does not drive it. In addition to providing in-memory access to needed database pages, the storage layer speeds up query and update processing with prefetch, read-ahead, shared scans, and write-behind.

The storage layer may also provide automatic maintenance of non-clustered indexes. This design only applies to centralized systems or to parallel systems with local indexes, i.e., partitions of indexes aligned

with the partitions of the table. This design forces row-by-row maintenance, although index-by-index maintenance can be required for correctness (certain updates of unique indexes) or for performance (sorting large sets of changes as appropriate for each index).

In very traditional designs, the storage layer may provide the navigation from non-clustered index to clustered index during query execution, but coupling these accesses inhibits many beneficial techniques such as covering a query with a non-clustered index alone, sorting references obtained from a non-clustered index, index intersection for conjunctive predicates and index union for disjunctive predicates, joining non-clustered indexes of the same table to cover a query not covered by any one index, and joining non-clustered indexes of two tables as a form of semi-join reduction.

Concurrency Control

Concurrency control is a very important service provided by the storage layer, both latching to protect in-memory-data structures from conflicting threads and locks to protect database contents from conflicting transactions. Alternatives to locking (“pessimistic concurrency control”) include validation (“optimistic concurrency control”) and versioning (“multi-version concurrency control”). Transactional memory may become an alternative to latching.

Locking and latching are described in detail elsewhere.

Logging and Recovery

Logging and recovery are also very important services provided by the storage layer, including transaction rollback, crash recovery, and media reconstruction. Since the units of recovery cannot be larger than the units of concurrency control, a high-concurrency system with key value locking or row-level locking cannot rely on page-based recovery provided by, for example, the file system or a network-attached storage service. The storage layer may provide log-shipping or continuous log-based replication using a “hot stand-by” database copy perpetually in recovery.

Logging and recovery are described in detail elsewhere.

Utilities

The broad term “utilities” covers all those operations needed for a complete database management system

product but not directly associated with query processing and transaction processing. Examples include index creation and removal, defragmentation and other forms of reorganization, moving and partitioning data, backup and restore, statistics creation and update, consistency checks and repairs, etc. Catching up on deferred maintenance can apply to materialized views, indexes, statistics, caches, and replicas.

Utilities may be offline, online, i.e., permitting user transactions to read and modify database data while the utility is scanning or reorganizing them, or incremental, i.e., the utility operation's effects such as index creation become useful to user transactions in multiple discrete steps.

As many utility operations move data similar to a query execution plan, and since similar services are required such as memory management, partitioning and pipelining for parallel execution, etc., many utilities can be implemented using the query processing component.

Key Applications

A storage manager is useful in many applications, practically all applications that map collections of records to pages on persistent storage. This includes entertainment software such as music players, personal productivity applications such as e-mail clients, and server-side data management applications such as mail servers and database management systems. A storage manager that provides buffering and transactions for both records and large fields (such as pictures, sound tracks, videos, messages, and documents) is even more widely useful.

Future Directions

While the basics of access methods, buffer pool management, concurrency control and recovery are all well understood and documented, the need for further development continues.

Queuing

Some database management systems have integrated queuing into their feature set. It enables access patterns typical for workflow applications, electronic mail, and service-oriented architectures. Technical challenges include hotspots for both insertion and deletion as well as transaction semantics that link data records into messages and "conversations" among automatic processes and human users.

XML Support

XML is not only a message format but also a storage format, in particular for human-authored documents and in service-oriented architectures based on message passing. XML in databases creates challenges for storage, compression, fine-grained concurrency control and recovery, consistency enforcement and verification, and query processing. While initial research prototypes and even commercial implementations exist, their optimization and adaptation for large applications is not yet complete.

Transactional Memory

Forthcoming many-core processors require, for performance and power-efficiency, data structures and algorithms that permit very high degrees of parallelism as well as appropriate concurrency control among concurrent software threads. Transactional memory is a promising approach to these issues. Hardware-assisted transactional memory enables very fast execution of critical sections as well as guaranteed success based on automatic rollback and re-execution.

Self-Tuning, Self-Repair, Total Cost of Ownership

Perhaps the greatest challenge in the design and implementation of a storage layer is the total cost of ownership, i.e., the amount of human attention and trouble-shooting required to ensure the desired levels of availability, integrity, and performance. For example, can the storage layer software prevent data loss unless a user knowingly and deliberately accepts a risk? Such a storage layer would have to require, during initial deployment, that multiple storage devices with independent failures be specified, among many other things. Similarly, could the storage layer software assume all responsibility for tuning the set of indexes, or could there be a standard interface to other relevant components such as the relational layer within a relational database?

Cross-references

- [Autonomic Computing](#)
- [B-Tree](#)
- [B-Tree Locking](#)
- [Buffer Pool](#)
- [Concurrency Control and Recovery](#)
- [Indexing](#)
- [Self-Tuning Database Systems](#)
- [Transaction](#)

Recommended Reading

1. Carey M.J., DeWitt D.J., Franklin M.J., Hall N.E., McAuliffe M.L., Naughton J.F., Schuh D.T., Solomon M.H., Tan C.K., Tsatalos O. G., White S.J., and Zwilling M.J. Shoring up persistent applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 23(2):383–394, 1994.
2. Chamberlin D.D., Astrahan M.M., Blasgen M.W., Gray J., King W.F., Lindsay B.G., Lorie R.A., Mehl J.W., Price T.G., Putzolu G.R., Selinger P.G., Schkolnick M., Slutz D.R., Traiger I.L., Wade B.W., and Yost R.A. A history and evaluation of system R. Commun. ACM, 24(10):632–646, 1981.
3. Härdter T. and Reuter A. Principles of transaction-oriented database recovery. ACM Comput. Surv., 15(4):287–317, 1983.
4. Hellerstein J.M., Stonebraker M., and Hamilton J.R. Architecture of a database system. Found. Trends Databases, 1(2):141–259, 2007.
5. Stonebraker M. Retrospection on a database system. ACM Trans. Database Syst., 5(2):225–240, 1980.

protocol is still effectively used on top of such new network technologies.

Foundations

In conventional IT systems, storage devices such as disk drives, disk arrays, tape drives and tape libraries are connected only to a single server. Such an IT system is often referred to as a server-centric system, in which a storage device is considered a dedicated peripheral device of a server to which the storage device is connected. Small Computer System Interface (SCSI) is the main technology of network infrastructure used in the server-centric system. Storage devices are connected together to a server by SCSI bus cables and provide block-level I/O services to the server using the SCSI protocol. The scalability of such bus technologies is rather limited. A single SCSI cable can be at most 25 m long, and allows a maximum of 15 storage devices to be connected. Storage space which a single server can accommodate is thus severely limited. A server is not able to directly access storage devices connected to another server. Instead the server has to access such storage devices indirectly through a Local Area Network (LAN). The inflexibility of interconnection thus scatters and duplicates data management functions among multiple servers. Storage management may be optimized within each server, but should be far from global optimization of the entire system. Such issues were not visible when expensive microprocessors limited the system capability. However, recent technology innovations have been decreasing the cost of high-speed microprocessors and explosively expanding the volume of managed digital data. The system capability is then more likely to be limited by the storage system, so the issues of the conventional server-centric systems have become more obvious.

In contrast, the innovations of network technologies have given new possibilities of directly transferring data between storage devices and servers and among storage devices. All servers connected to a storage network are able to access all storage devices in the same network. Such a flexible connection helps storage devices to be consolidated in the network, thus isolating storage resource management from the server. That is, storage resources are placed and managed within a storage network, and servers are then positioned around the network. This type of IT system is often referred to as a storage-centric system. [Figure 1](#) illustrates a server-centric system and a storage-centric system.

Storage Network Architectures

KAZUO GODA

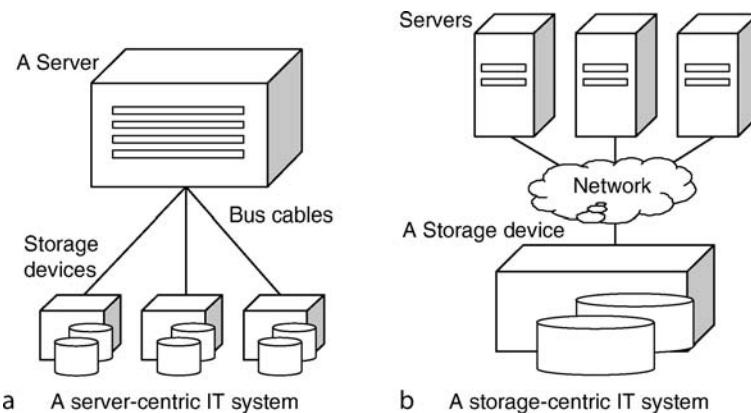
The University of Tokyo, Tokyo, Japan

Definition

Storage Network Architecture is the conceptual structure and logical organization of a network whose main purpose is to transfer data between storage devices and servers and among storage devices. The term Storage Network is identified with such a network, but is sometimes used to refer to a storage system communicating over a network. Related terms such as Storage Area Network and Network Attached Storage are described in separate entries. Note that usages of terms related to storage network architectures may depend on contexts at times.

Historical Background

The first version of SCSI was released in 1986. SCSI then became deployed in many open systems, thus acquiring the position of the standard IO technology. However, after Fibre Channel technology was invented in the late 1990s, Fibre Channel rapidly extended its use in the market. Recent mid-range and high-end storage systems have deployed Fibre Channel as the standard IO technology. Alternative storage network technologies such as iSCSI may be available in entry-level systems. The SCSI bus technology has been replaced with Fibre Channel and iSCSI, but the SCSI



Storage Network Architectures. Figure 1. A server-centric IT system and a storage-centric IT system.

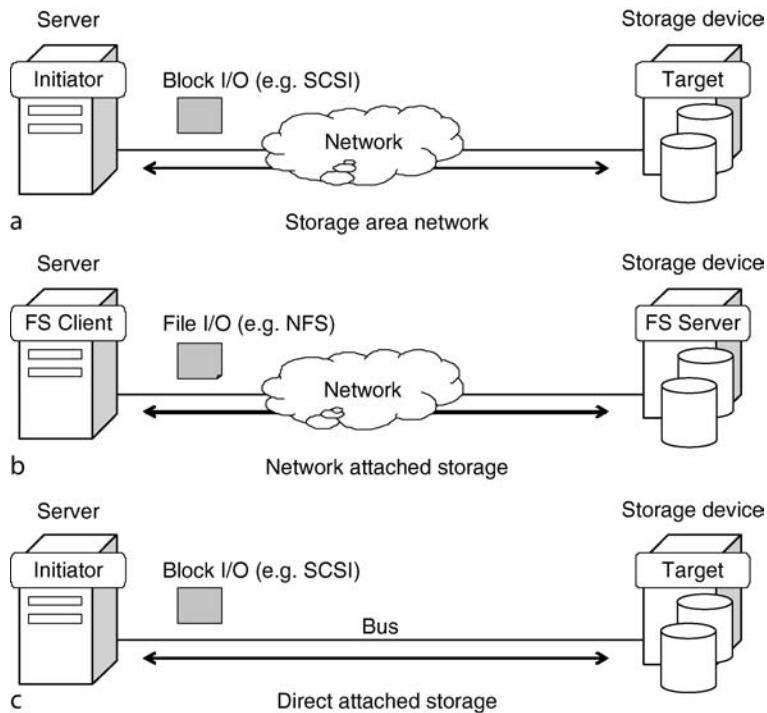
Storage networks have given system designers several alternatives for linking storage devices and servers. Currently available interconnections can be grouped into three architectures: Storage Area Network, Network Attached Storage and Direct Attached Storage. This categorization is widely accepted although it may not be formal. Figure 2 presents an illustrative comparison of these architectures. Separate entries give a formal definition of each storage network architecture.

A Storage Area Network, which is often abbreviated to SAN, is a network which mainly transfers data from/to storage devices. A SAN is used to connect storage devices with servers and with other storage devices. A storage system which is implemented over a network is sometimes referred to as a SAN too. A SAN provides block-level I/O services. Most SANs are implemented on top of Fibre Channel technology, although alternative technologies such as internet SCSI (iSCSI) are available for SANs. A SAN which is implemented on Fibre Channel technology is sometimes referred to as Fibre Channel SAN.

Fibre Channel is a gigabit-level network technology, which is mostly used for SANs at present. The Fibre Channel protocol stack is divided into two parts. The lower part defines fundamental network infrastructure, which can be further subdivided into four layers (FC-0–FC-3). FC-0 defines transmission media such as optical/electrical interfaces, cables and connectors. FC-1 specifies signal encoding and decoding (8b/10b conversion and 40b ordered sets) and link controls. FC-2 defines frame formats, frame transmission management (sequences and exchanges) and flow controls. FC-3 defines common services such as multipathing. On top of these fundamental layers, the

higher part (FC-4) defines protocol mappings to application protocols. When Fibre Channel is used for SANs, Fibre Channel Protocol for SCSI (FCP) maps the Fibre Channel infrastructure to the SCSI protocol. Fibre Channel is characterized by its powerful transmission capabilities such as serial transmission, low error rate and low latency. Processing of the Fibre Channel protocol is usually implemented in host bus adapter (HBA) cards at the hardware level so as to relieve servers' processors. These attributes are preferable for SANs, however, they increase the cost of Fibre Channel network devices. Fibre Channel SANs are thus deployed mainly in mid-range and larger IT systems. On the other hand, iSCSI is an approach to exploit the IP network infrastructure so that SANs can be installed and operated at much lower cost. iSCSI, which is placed on the top of the TCP/IP protocol stacks, encapsulates SCSI data in TCP/IP packets. At present, iSCSI is used for connecting servers and entry-level storage systems. Internet Fibre Channel Protocol (iFCP) and Fibre Channel over Internet Protocol (FCIP) are alternative approaches to use TCP/IP technology for SANs. These protocols can transmit Fibre Channel frames over IP networks. In contrast to iSCSI, iFCP and FCIP are mainly used for connecting remotely distant SAN islands. Note that Fibre Channel and iSCSI are network technologies that only replace the classical SCSI bus technology. The SCSI protocol is still utilized on top of such network technologies even in today's storage-centric systems.

Network Attached Storage is a storage device that is connected to a network and provides file access services. Network File System (NFS) and Common Internet File System (CIFS) are two major protocols used



Storage Network Architectures. Figure 2. Three storage network architectures.

for NAS networks. That is, NAS is used over IP network technology.

The history of NAS can be traced back to file sharing functions provided by operating systems. NFS and CIFS were developed for sharing files between servers in the conventional server-centric system. A file server, which exports file access services to other computers, is a type of implementation of a NAS device. Recent NAS devices are comprised of dedicated hardware and software because of the increasing demand on reliability and performance. A diskless NAS device, specifically a NAS device which has only controllers but contains no disk drives, is sometimes referred to as a NAS gateway or a NAS head. A NAS gateway/head can provide NAS clients with file access services to other storage devices that only export block-level I/O services. That is, a NAS gateway/head could be considered a service bridge between a SAN and a NAS network. Since NAS systems are based on TCP/IP technology, poor access performance is typically observed in comparison with a SAN. However, NAS systems have the strong benefit of exploiting existing IP network resources. The cost effectiveness of NAS systems has expanded their use especially in the entry-level markets.

The conventional storage system architecture, in which storage devices are connected to a single server via a SCSI bus cable, has been renamed to Direct Attached Storage after new storage network architectures such as SAN and NAS appeared. Direct Attached Storage is often abbreviated to DAS.

Key Applications

The flexible interconnection provided by storage networks enables storage devices to communicate with each other. In such a system, not necessarily all the functions need to be executed on server processors. Instead, executing some software codes on storage devices may be more efficient. Actually, storage developers are accommodating different applications into their storage devices. Below are described major storage network applications.

LAN-free Backup: in the server-centric storage architecture, a dedicated server connected to a LAN is usually responsible for creating and managing backup copies. The backup server reads data from other servers through the LAN and writes the data to the archiving storage, such as disk arrays and tape libraries, connected to the backup server. In contrast, the storage-centric architecture enables all the servers to access all the

storage devices. The backup server can thus copy data directly from the source storage devices to the archiving storage device. Such LAN-free backup can be considered an approach of moving the copy traffic from the LAN toward the SAN.

Server-free Backup: server-free backup is a more advanced solution, in which storage devices or network devices connected in a storage network directly make a backup copy. Thus, the storage network does manage backup copies without any dedicated backup servers. A copy function which is incorporated in storage devices and/or network devices is often referred to as third-party copy.

Remote Replication: remote replication, which can be seen as a type of server-free backup, keeps a fresh backup copy in a remote data center. Communication between a local data center and a remote data center usually involves non-negligible latency. Two techniques are used for remote replication. Synchronous remote replication forwards a given write command to a remote storage device and then commits the command after the forwarded command is acknowledged by the remote device. This strict mode can synchronize data between the two storage devices all the time; no data would be lost even if a severe disaster damages the local storage device. However, high communication latency between the devices is likely to affect the response time of write commands, thus degrading application performance. In contrast, asynchronous remote replication commits a write command without waiting for the command to be acknowledged by the remote device. Inter-device communication latency would be invisible, but data coherence could not be guaranteed.

Data Sharing and Code Conversion: in a storage-centric environments, a storage device is shared by multiple servers. Sharing the data among multiple servers is also a natural approach. However, presentation forms of data usually depend on microprocessor architectures and operating systems. A file written by a server to a storage device may not be directly interpreted by another server. In the conventional server-centric system, the dedicated application running on the server converts data formats and character codes so that the application can interpret the data appropriately. Such code conversion facilities are being implemented in storage infrastructure. That is, when a server tries to read data stored in a storage system, the storage system converts and then exports the data to the server. Code conversion functions implemented in storage

networks are deployed for downsizing from mainframes towards open systems and for file sharing between different types of machines.

SAN File system: a SAN file system is a file system which exports services for accessing files stored in storage devices connected to a SAN. A volume of a SAN file system is often shared among multiple servers. Thus, concurrency control is a key technology for a SAN file system. A SAN file system is deployed in many high-performance clusters and also in several high-end NAS systems.

Cross-references

- ▶ [Direct Attached Storage](#)
- ▶ [IP Storage](#)
- ▶ [Network Attached Storage](#)
- ▶ [SAN File System](#)
- ▶ [Storage Area Network](#)
- ▶ [Storage Management](#)

Recommended Reading

1. Benner A.F. Fibre channel for SANs. McGraw-Hill Professional, 2001.
2. Clark T. IP SANS: a guide to iSCSI, iFCP, and FCIP protocols for storage area networks. Addison-Wesley Professional, Reading, MA, 2001.
3. Clark T. Designing storage area networks: a practical reference for implementing fibre channel and IP SANs. Addison-Wesley, Reading, MA, 2003.
4. Robert W., Kembel R. W., and Cummings R. The fibre channel consultant: a comprehensive introduction. Northwest Learning Association, 1998.
5. Storage Network Industry Association. The Dictionary of Storage Networking Terminology. Available at: <http://www.snia.org/>
6. Troppens U., Erkens R., and Müller W. Storage networks explained. Wiley, London, 2004.

S

Storage Networking Industry Association

HIROSHI YOSHIDA
Fujitsu Limited, Yokohama, Japan

Synonyms

[SNIA](#)

Definition

A non-profit trade association dedicated to the development and promotion of standards, technologies,

and educational services, to empower organizations in the management of information.

Key Points

The SNIA works toward these goals by forming and sponsoring technical work groups, producing a series of conferences, building and maintaining a vendor neutral technology center, and promoting activities that expand the breadth and quality of the storage and information management market. With seven regional affiliates spanning the globe, SNIA represents the voice of the storage industry on a worldwide scale.

Cross-references

- ▶ Storage Management
- ▶ Storage Management Initiative-Specification

Recommended Reading

1. Storage Network Industry Association, Storage Network Industry Association, 2007. <http://www.snia.org/>

access strategies. HEAVEN (Hierarchical Storage and Archive Environment for Multidimensional Array Database Management Systems) is specifically designed and optimized for storing multidimensional array data on tertiary storage media.

Historical Background

Large-scale scientific experiments or supercomputing simulations often generate large amounts of multidimensional data sets. Data volume may reach hundreds of terabytes (up to petabytes). Typically, these data sets are permanently stored as files in an archival mass storage system, on up to thousands of magnetic tapes. Access times and/or transfer times of these kinds of tertiary storage devices, even if robotically controlled, are relatively slow. Nevertheless, tertiary storage systems are currently for the common state of the art storing such large volumes of data, because magnetic tapes are much cheaper than hard disk devices. This will also be the future trend. Furthermore, tapes are a good example for Green-IT. The generation of new data will increase extremely, because of new satellites, sensors, parameters etc. Consequently, scientists need more and more capacity for storing these large amounts of data, and tapes are well prepared for this task.

Concerning data access in the High Performance Computing (HPC) area, the main disadvantages are high access latency compared to hard disk devices and to have no random access. A major bottleneck for scientific applications is the missing possibility of accessing specific subsets of data. If only a subset of such a large data set is required, the whole file must be transferred from tertiary storage media. Taking into account the time required to load, search, read, rewind, and unload several cartridges, it can take many hours to retrieve a subset of interest from a large data set. Entire files (data sets) must be loaded from the magnetic tape, even if only a subset of the file is needed for further processing.

Furthermore, processing of data across a multitude of data sets, for example, time slices, is hard to support. Analysis of dimensions been contrary to storage patterns and requires network transfer of each required data set, implying a prohibitively immense amount of data to be shipped. Another disadvantage is that access to data sets is done on an inadequate semantic level. Applications accessing HPC data have to deal with directories, file names, and data formats instead of accessing multidimensional data in terms of area of

Storage of Large Scale Multidimensional Data

BERND REINER¹, KARL HAHN²

¹Technical University of Munich, Munich, Germany

²BMW AG, Munich, Germany

Synonyms

Hierarchical storage management; HSM; Multidimensional database management system; Raster data management

Definition

An identified major bottleneck today is fast and efficient access to and evaluation of high performance computing results. This contribution addresses the necessity of developing techniques for efficient retrieval of requested subsets of large datasets from mass storage devices (e.g., magnetic tape). Furthermore, the benefit of managing large spatio-temporal data sets, e.g., generated by simulations of climate models or physical experiments, with Data Base Management Systems (DBMS) will be shown. Such DBMS must be able to handle very large data sets stored on mass storage devices. This means DBMS need a smart connection to tertiary storage systems with optimized

interest and time interval. Examples of large-scale HPC data are climate-modeling simulations, cosmological experiments and atmospheric data transmitted by satellites. Such natural phenomena can be modeled as spatio-temporal array data of some specific dimensionality. Their common characteristic is that a huge amount of Multidimensional Discrete Data (MDD) has to be stored. For overcoming the above mentioned shortcomings, and for providing flexible data management of spatio-temporal data, HEAVEN (Hierarchical Storage and Archive Environment for Multidimensional Array Database Management Systems) was implemented [5].

Foundations

In order to implement smart management of large-scale data sets held on tertiary storage systems, HEAVEN combines the advantages of efficient retrieval and manipulation of data sets by using multidimensional array DBMS, and storing big amounts of data sets on tertiary storage media. This means the DBMS must be extended with easy to use functionalities to automatically store and retrieve data to/from tertiary storage systems without user interaction. A description of related work can be found in [5].

Such intelligent concepts are implemented within the European funded project ESTEDI, and integrated into the kernel of the multidimensional array DBMS RasDaMan (Raster Data Management). RasDaMan is designed for generic multidimensional array data of arbitrary size and dimensionality. In this context, generic means that functionality and architecture of RasDaMan are not tied to particular application areas.

Figure 1 depicts the architecture of HEAVEN.

One can see the original RasDaMan architecture with the RasDaMan client, RasDaMan server and the underlying conventional DBMS (e.g., Oracle, which is used as a storage and transaction manager). The additional components for the tertiary storage interface are the Tertiary Storage Manager (TS-Manager), File Storage Manager and Hierarchical Storage Management System (HSM-System). The TS-Manager and File Storage Manager are included in the RasDaMan server. The HSM-System is a conventional product like TSM/HSM (Tivoli Storage Manager) from IBM. Such an HSM-System can be seen as a normal file system with unlimited storage capacity. In reality, the virtual file system of HSM-Systems is separated into a limited cache on which the user works (load or store

his data), and a tertiary storage system with robot controlled tape libraries. The HSM-System automatically migrates or stages data to or from the tertiary storage media, if necessary.

Efficient Storage of Large Multidimensional Data

For overcoming the major bottleneck, i.e., the missing possibility of accessing specific subsets of data (MDD), the tiling concept of the RasDaMan DBMS is introduced. A MDD object consists of an array of cells of some base type (e.g., integer, float or arbitrary complex types), which are located on a regular multidimensional grid. An often discussed approach is chunking or tiling of large data [1,8,2]. Basically, chunking means the subdividing of multidimensional arrays into disjoint sub-arrays. Tiling is more general than chunking, because sub-arrays don't have to be aligned or have the same size. In RasDaMan, MDD can be subdivided into regular or arbitrary tiles. Consequently one MDD object is a set of multidimensional tiles. In RasDaMan, every tile is stored as one single Binary Large Object (BLOB) in the underlying relational DBMS. This makes it possible to transfer only a subset of large MDD from the DBMS (or tertiary storage media) to client applications, because access granularity is one single tile. Also, the problem of inefficient access to data sets stored according to their generation process order is not any longer relevant with the tiling strategy of RasDaMan. This will mainly reduce access time and network traffic. The query response time scales with the size of the query box, not any longer with the size of MDD.

Data Export to Tertiary Storage Media

The export of data sets to tertiary storage media is two-tiered. The first step is the migration of the data sets from RasDaMan to the cache area (RAID-System) of the HSM-System. Transferring data sets from the hard disk of the underlying DBMS of RasDaMan to a RAID-System is very fast. Within a second step the migration from the cache area of the HSM-System to the tertiary storage media takes place. This process is performed by the HSM-System, and does not concern the RasDaMan system regarding I/O workload. During this process RasDaMan can execute another export process or user request in parallel.

Data Retrieval

RasDaMan provides an algebraic query language RasQL, which extends SQL with powerful

multidimensional operators like geometric, induced and aggregation operators. The primary benefit of such a complex query language is the minimization of data transfer between database server and client. Areas of interest can be specified with geometric operators, and complex calculations can be executed on the server side. Only the result is transferred to the client instead of the entire object [6,7]. With this feature, RasDaMan, overcomes the mentioned shortcoming of processing data across a multitude of data sets, because RasDaMan transfers only a minimum of data to the client. Furthermore, the query language RasQL provides data access on an adequate semantic level. Users can formulate queries such as: "average temperature on the earth surface of altitude y in the area of latitude x and longitude z".

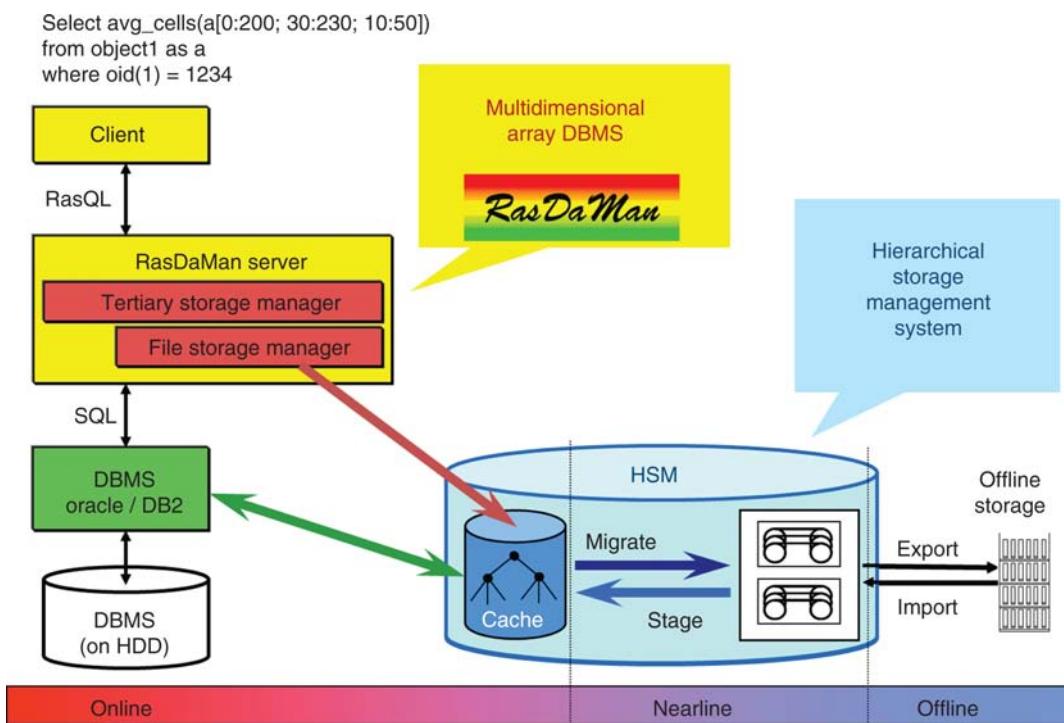
With respect to data accessibility, three well-defined areas can be differentiated, i.e., online, nearline and offline area (Fig. 1). Data sets stored online means that data sets are stored on hard disk and, therefore, access time is very fast. Data sets stored on magnetic tape and stored in robot controlled libraries are called nearline data. Access time is much higher than with online access, but the process of data retrieval is done

automatically. If data sets are stored in the offline area, user interaction is necessary for retrieving data sets (tapes are not robot controlled).

The new RasDaMan tertiary storage functionality is based on the TS-Manager module. If a query is executed, the TS-Manager knows (by metadata) whether the needed data sets are stored on hard disk (online area, DBMS or HSM-Cache) or on tertiary storage media. If the data sets are held on hard disk, the query will be processed very fast. If the data sets are stored on one or more tertiary storage media, the data sets must be imported into the database system (cache area for tertiary storage data) first. The import of data sets stored on tertiary storage media is done by the TS-Manager automatically whenever a query is executed and those data sets are requested. After the import process of the data sets is done, RasDaMan can handle the data sets in the normal way. The complexity of the RasDaMan storage hierarchy is completely hidden from the user.

Techniques for Reducing Tertiary Storage Access Time

The access time for tape systems is by order of magnitude slower than for hard disk devices. It is



Storage of Large Scale Multidimensional Data. Figure 1. Extended RasDaMan architecture with tertiary storage connection.

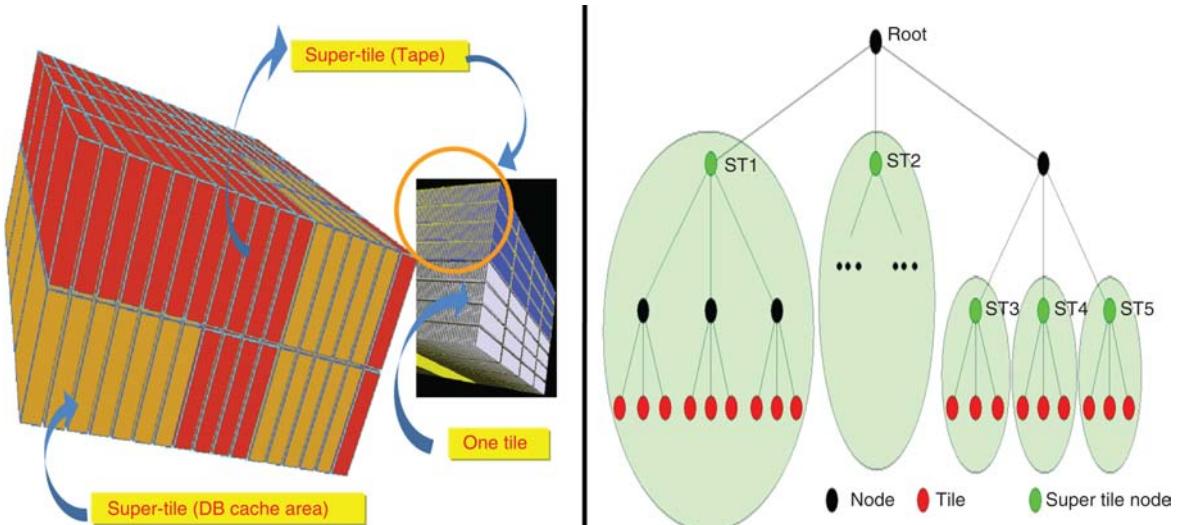
important to use data management techniques for the efficient retrieval of arbitrary areas of interest from large data sets stored on tertiary storage devices. Hence, techniques that partition data sets into clusters based on optimized data access patterns and storage device characteristics, has to be developed. Therefore, methods for reducing tertiary storage access time, i.e., Super-Tile concept, data clustering and data caching are presented [5,6].

Super-Tile Concept In RasDaMan, DBMS tiles (BLOBs) are the smallest unit of data access. Typical sizes of tiles stored in RasDaMan range from 64 KB to 1 MB and are optimized for hard disk access [2]. Those tile sizes are much too small for data sets held on tertiary storage media. It is necessary to choose different granularities for hard disks and tape access because they differ significantly in their access characteristics. Hard disks have fast random access, whereas tape systems have sequential access with much higher access latency. The average access time for tape systems (20–180 s) is by order of magnitude slower than for hard disk drives (5–12 ms), whereas the difference between transfer rate is not significant [5,10]. For this reason, HEAVEN exploits the good transfer rate of tertiary storage systems, preserving the advantages of the tiling concept. The main goal is to minimize the number of media load and search operations.

It is unreasonable to increase the RasDaMan MDD tile size, because then RasDaMan would loose the

advantage of reducing transfer volumes when accessing data on HDD. The solution is to introduce an additional data granularity as provided by the so-called Super-Tile. The main goal of the new Super-Tile algorithm is a smart combination of several small MDD tiles to one Super-Tile to minimize tertiary storage access costs. “Smart” means to exploit the good transfer rate of tertiary storage devices and to preserve advantages of other concepts like data clustering. The left side of Fig. 2 visualizes one three dimensional MDD with Super-Tile and tile granularity. An algorithm for computing Super-Tiles was developed which combines tiles of spatial neighborhood within the multidimensional object. For the realization, HEAVEN utilizes information of the RasDaMan R+ tree index [3,5].

The creation of the multidimensional index and the index access is no performance issue compared to data retrieval and data processing. Also, the primary criticism of the R-tree, that performance problems for very many dimensions occur, is not relevant for the application field. The used scientific data from various scientific fields (e.g., climat-modeling simulations, cosmological experiments, atmospheric data, earth observation, computational fluid dynamics) does not have more than five dimensions. Therefore, the integration of advanced multidimensional index methods, e.g., the bitmap index for scientific data proposed by Rishi Sinha and Marianne Winslet [9] or the UB tree proposed by Rudolf Bayer [4], was not considered.



Storage of Large Scale Multidimensional Data. Figure 2. Left: Visualization of one MDD with Super-Tile and Tile granularity. Right: Example R+ tree index of one MDD with Super-Tile nodes.

The conventional R+ tree index structure of the multidimensional DBMS was extended to handle such Super-Tiles stored on tertiary storage media. This means that information (whether tiles are stored on hard disk or on tertiary storage media) must be integrated into the index. Tiles of the same sub index of the R+ tree are combined into a Super-Tile and stored within a single file on tertiary storage medium (see right side of Fig. 2). Super-Tile nodes can exist on arbitrary levels of the R+ tree. Super-Tiles are the access (import/export) granularity of MDD on tertiary storage media, which preserve the advantages of the RasDaMan tiling concept (load minimum data) and exploit the good transfer rates of tertiary storage devices. More details about determining optimal file sizes on tertiary storage media can be found in [5].

Clustering Clustering is particularly important for tertiary storage systems where positioning time of the device is very high. The main goal is to minimize the number of search and media load operations and to reduce the access time of clusters read from tertiary storage system when subsets are needed. Clustering exploits the spatial neighborhood of tiles within data sets. Clustering of tiles according to spatial neighborhood on one disk or tertiary storage system, proceed one step further in the preservation of spatial proximity, which is important for the typical access patterns of array data, because users often request data using range queries, which implies spatial neighborhood.

The R+ tree index used to address tiles already defines the clustering of the stored MDD. With the developed Super-Tile concept intra Super-Tile clustering and inter Super-Tile clustering can be distinguished. The implemented algorithm for computing Super-Tiles maintains the predefined clustering of subtrees (of Super-Tile nodes) of the R+ tree index and achieves intra Super-Tile clustering (left side of Fig. 2). The export algorithm (export of Super-Tiles to tertiary storage) implements the inter Super-Tile clustering within one MDD. Super-Tiles of one MDD are written to tertiary storage media in the clustered order (pre-defined R+ tree clustering).

Caching In order to reduce expensive tertiary storage media access, the underlying DBMS of RasDaMan is used as a hard disk cache for data sets held on tertiary storage media. The general goal of caching tertiary storage data (Super-Tile granularity) is to minimize

expensive loading, rewinding and reading operations from slower storage levels (e.g., magnetic tape). In the tertiary storage version of RasDaMan, requested data sets held on tertiary storage media are migrated to the underlying DBMS of RasDaMan (Fig. 1). The migrated Super-Tiles are now cached in the DBMS. After the migration, the RasDaMan server transfers only requested tiles from the DBMS to the client application.

The tertiary storage Cache-Manager evicts data (Super-Tile granularity) from the DBMS cache area only if necessary, i.e., the upper limit of cache size is reached. A special H-LRU (HEAVEN Least Recently Used) algorithm was developed, and together with the caching component of the HSM-System a caching hierarchy (Fig. 1) was built.

Conclusion

The main goal was the realization of optimized management of large-scale data sets stored on tertiary storage systems combined with access functionality like retrieval of subsets. Therefore, a multidimensional array DBMS for optimized storage, retrieval and manipulation of large multidimensional data was introduced. In order to handle hundreds of petabytes stored on tertiary storage media, an interface was presented connecting tertiary storage systems to the multidimensional array DBMS RasDaMan. The Hierarchical Storage and Archive Environment for Multidimensional Array Database Management Systems (HEAVEN) is specifically designed and optimized for storing multidimensional array data on tertiary storage media. For this reason, the query response time scales with the size of the query box and not with the size of the multidimensional data. This will dramatically reduce access time compared with the traditional access case.

Key Applications

HEAVEN is specifically designed for storing large-scale multidimensional array data (hundreds of terabytes) on tertiary storage media, and is optimized toward HPC. Addressed HPC areas are for example climate-modeling simulations, cosmological experiments and, atmospheric data transmitted by satellites.

Cross-references

- [Archiving Experimental Data](#)
- [Clustering](#)
- [Data Partitioning](#)

- ▶ Database Management System
- ▶ Disk
- ▶ Query Language
- ▶ Raster Data Management and Multi-Dimensional Arrays
- ▶ Rtree
- ▶ Storage Management

Recommended Reading

1. Chen L.T., Drach R., Keating M., Louis S., Rotem D., and Shoshani A. Efficient organization and access of multi-dimensional datasets on tertiary storage. *Inf. Syst.*, 20(2), 1995.
2. Furtado P.A. Storage Management of Multidimensional Arrays in Database Management Systems, PhD Thesis, Technische Universität München, 1999.
3. Gaede V. and Günther O. Multidimensional access methods. *ACM Comput. Surv.*, 30(2):170–231, 1998.
4. Ramsak F., Markl V., Fenk R., Zirkel M., Elhardt K., and Bayer R. Integrating the UB-Tree into a Database System Kernel. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 263–272.
5. Reiner B. HEAVEN – A Hierarchical Storage and Archive Environment for Multidimensional Array Database Management Systems, PhD Thesis, Technische Universität München, 2005.
6. Reiner B., Hahn K., Höfling G., and Baumann P. Hierarchical Storage Support and Management for Large-Scale Multidimensional Array Database Management Systems. In Proc. 13th Int. Conf. Database and Expert Syst. Appl., 2002, pp. 689–700.
7. Ritsch R. Optimization and Evaluation of Array Queries in Database Management Systems, PhD Thesis, Technische Universität München, 1999.
8. Sarawagi S. and Stonebraker M. Efficient organization of large multidimensional arrays. In Proc. 10th Int. Conf. on Data Engineering, 1994, pp. 328–336.
9. Sinha R.R. and Winslett M. Multi-resolution bitmap indexes for scientific data. *ACM Trans. Database Syst.*, 32(3):2007.
10. Yu J. and DeWitt D. Processing satellite images on tertiary storage: a study of the impact of tile size on performance. In Proc. 5th NASA Goddard Conf. on Mass Storage Systems and Technologies, 1996.

Storage Power Management

KAZUO GODA
The University of Tokyo, Tokyo, Japan

Definition

Storage Power Management is a process of improving the efficiency of electric power consumption of all the concerned storage resources including storage devices, storage controllers and storage network devices. Storage Power Management may sometimes cover related

equipment such as power supplies and cooling apparatuses. The definition of “the efficiency” may depend on the situation; system designers and administrators often need to balance electric power reduction against performance degradation.

Historical Background

Electric power consumption of storage devices was discussed mainly with regard to battery-operated computing environments such as laptop PCs. However, due to the rapid growth of power consumption in data centers and the increased interest in environmental issues, much attention has recently been paid to electric power consumption of enterprise-level storage systems. Energy efficiency is recognized as a new direction for research and development of storage systems.

Foundations

Hard disk drives are main components of modern storage systems. In the disk drive, a spindle motor, which rotates metal platters at high speed, consumes most of the electric power. Gurumurthi et al. [3] report that the spindle motor can account for 81% of the power consumed by the disk. For a given disk drive, the power consumption of its spindle motor P theoretically relates to the angular velocity ω as

$$P = \frac{K_e^2 \omega^2}{R}$$

where K_e is a motor voltage constant and R is a motor resistance constant. In reality, since the spindle motor rotates the platters against air drag, the angular velocity ω may have a cubic or greater effect on the power consumption P . Disk drive manufacturers have increased the rotational speed to decrease the access latency and improve the transfer rate. Disk array developers have been accommodating a number of such high-speed disk drives into a single disk array enclosure. Thus, greater electric power consumption is often seen in enterprise-level storage systems.

Many commercial disk drives have a “stand-by” mode. While a disk drive is in the stand-by mode, its head is unloaded from the platters to the ramp and its spindle motor is completely suspended. The disk drive consumes much less electric energy in the stand-by mode than in the active (currently processing read/write requests) and idle (being able to start processing read/write requests immediately) modes. However, the transition to/from the stand-by mode involves

non-negligible overhead of time and electric energy. Especially, spinning up a spindle motor to regular speed takes several to tens of seconds and consumes tens to hundreds of joules. Such a significant overhead associated with low-power modes is rarely seen in other computer components including processors and memory. Several commercial disk drives may have different low-power modes other than the stand-by mode. For example, a disk drive in a “low-rpm” mode may keep rotating the platters at lower speed with its head unloaded, accordingly consuming more power but involving less overhead.

The minimum time that the disk needs to be idle for the power saving achieved to exceed the control energy overhead is called “break-even time.” The break-even time is a specific parameter of a disk drive. Assuming that the system could perfectly predict accesses that would be issued to a disk drive in the future, the system could spin down the disk drive after the disk becomes idle only if the idle period will be longer than its break-even time. In turn, the system could also spin up the drive in advance before new disk accesses are issued. This “oracle power management” gives the maximum possible energy saving for a given series of disk accesses. However, predicting the future disk accesses perfectly is impossible in reality. Alternative solutions have been studied so far.

Typical techniques of disk power management are based on idleness threshold. The simplest strategy is to spin down a disk drive to a low-power mode after a predetermined time has elapsed since the last disk access. This is based on the heuristic prediction that a disk drive is likely to continue to be idle if it has been idle for a long period. This traditional strategy is deployed in many commercial low-end disk drives, since it works effectively in end-user computing environments where the workload is dominated by interactive applications and users can accept reasonable spinning-up latency. More sophisticated techniques that try to tune the idleness threshold adaptively have been also investigated.

Dynamic Rotations Per Minute (DRPM) [3] is an attempt to exploit innovative disk drives that have the capability of dynamically changing the rotational speeds. Instead of completely spinning down the disk drive, this idea controls the rotational speeds adaptively by observing disk access performance. DRPM is helpful to balance between power and performance tradeoffs more flexibly compared with the conventional low-power modes.

Gurumurthi et al. [3] validated potential benefits of DRPM techniques on a simulator. At present, disk drives that can change their rotational speeds are not commercially available but merely reported in papers.

In enterprise systems, a number of individual disk drives are incorporated into a disk array and managed by the array controller. Data layout among the disk drives is a key to power management of such systems. A disk array which has the capability of spinning down member disk drives is called Massive Array of Idle Disks (MAID). The original paper [2] of MAID investigated the caching strategy. Suppose that member disk drives of a given disk array can be divided to a small number of active disk drives and a large number of passive disk drives, and all the blocks that are exported to the server are originally located in the passive disk drives. By replicating hot blocks that are frequently accessed onto the active disks, the array can achieve long idle periods for the passive disks so that they may be spun down. Another paper [1] studied a block migration strategy called Popular Data Concentration (PDC), which clusters hot blocks onto particular disk drives in order to spin down the other drives.

This entry focuses on electric power consumption of disk drives, which are main components of modern storage systems. Other components such as RAID controllers and power supplies and other types of storage devices such as optical discs, electromagnetic tapes and solid-state memory should be more carefully studied in the future.

Key Applications

The rapid growth of electric power consumption has stimulated the economic demand of energy saving technologies. In addition, a variety of government-level restrictions and business-level standards are being considered to resolve or mitigate environmental issues. Much more attention is likely to be paid to Storage Power Management in the future.

Cross-references

- ▶ [Deduplication](#)
- ▶ [Disk Power Saving](#)
- ▶ [Massive Array of Idle Disks](#)

Recommended Reading

1. Carrera E.V., Pinheiro E., and Bianchini R. Conserving disk energy in network servers. In Proc. 17th Annual Int. Conf. on Supercomputing, 2003, pp. 86–97.

2. Colarelli D. and Grunwald D. Massive arrays of idle disks for storage archives. In Proc. 16th Annual Int. Conf. on Supercomputing, 2002, pp. 1–11.
3. Gurumurthi S., Sivasubramaniam A., Kandemir M., and Hubertus F. Reducing disk power consumption in servers with DRPM. IEEE Comput 36(12):59–66, 2003.
4. Hitachi Global Storage Technologies, Inc. Quietly Cool. White Paper, 2004.
5. Lu Y.-H. and Micheli G.D. Comparing System-Level Power Management Policies. IEEE Design Test Comput, 8(2):10–19, 2001.

Storage Protection

KENICHI WADA

Hitachi Limited, Tokyo, Japan

Synonyms

Data protection

Definition

Storage protection is a kind of data protection for data stored in a storage system. The stored data can be lost or becomes inaccessible due to, mainly, a failure in storage component hardware (such as a hard disk drive or controller), a disastrous event, an operator's mistake, or intentional alteration or erasure of the data.

Storage protection provides the underlying foundation for high availability and disaster recovery.

Historical Background

In 1956, IBM shipped the first commercial storage that had a hard disk drive. To protect data from bit errors on disk platters, the hard disk drive commonly uses cyclic redundancy check (CRC) and an error-correcting code (ECC).

CRC and ECC cannot protect data from a whole disk failure in which an entire disk becomes inaccessible (for example, because of a disk head crash). The IBM 3990, which was shipped in the 1980s, had the replication functionality in which two identical copies of data were maintained on separate media. This approach protected data from this kind of failure. Replication functionality can be implemented in many other layers of the computer system. Most DBMS support database replication. Some file systems and Logical Volume Managers have file or volume replication functionality. Further, many storage systems and storage

virtualization appliances support volume replication functionality.

RAID (Redundant Array of Inexpensive Disks) is another technology for protecting data from whole disk failure. D. Patterson et al. published a paper "A Case for Redundant Arrays of Inexpensive Disks (RAID)" in June 1988 at the SIGMOD conference [6]. This paper introduced a five level data protection scheme. The term RAID was adopted from this paper, but currently RAID is an acronym for *Redundant Arrays of Independent Disks*. It is noted that the patent covering RAID level 5 technology was issued in 1978 [5].

RAID level 1 is a kind of replication. RAID level 2 to 5 can reduce the capacity required to protect data against disk drive failure than replication, but it is limited to protect disk drive failure. Replication, on the other hand, can be used to protect databases, file systems and logical volume. Further replication can be used for disaster recovery, if data are replicated remotely.

Foundations

Hard disk drives commonly use Reed-Solomon code [7] to correct bit errors. Data in hard disk drives is usually stored in fixed length blocks. Controllers in hard disk drives calculate ECC for each block and record it associated with the original data. When data are read, the controller checks data integrity using ECC. CRC can be used with ECC for detecting bit errors and/or reducing the possibility of correction error.

Most DBMS support database replication with master/slave relation between the original and the replica. The master process updates and transfer it to the slave. This type of replication can provide high availability to the client of the DBMS in case of storage system failures as well as server failures. Another type of database replication is multi-master, which is mostly used to provide high performance parallel processing. Both types can be either synchronous or asynchronous replication. In synchronous replication, updates made in original are guaranteed in the replica, note there may be some delay in asynchronous replication.

Volume replication by storage system is also widely accepted as data protection. There are synchronous and asynchronous replications, the same as database replication. Asynchronous volume replication is often used for long distance remote replication. It may

prevent performance degradation caused by replication delay, but could cause some data loss in case of recovery. Synchronous replication, on the other hand, may provide no data loss recovery, but may cause performance degradation due to replication delay. Volume replication is also used within a local datacenter for online backup. Backup servers use replica volume for backup during original volume is online. To support this, a storage system can pause update delegation from original to replica volume.

RAID (Redundant Array of Independent Disks) is a set of disks from one or more commonly accessible disk subsystems, combined with a body of control software, in which part of the physical storage capacity is used to store redundant information about user data stored on the remainder of the storage capacity. The term *RAID* refers to a group of storage schemes that divide and replicate data among multiple disks, to enhance the availability of data at desired cost and performance levels. A number of standard schemes have evolved which are referred to as *levels*. Originally, five RAID levels were introduced [6], but many more variations have evolved. Currently, there are several sublevels as well as many non-standard levels. There are trade-offs among RAID levels in terms of performance, cost and reliability.

Key Applications

Storage protection is essential to achieve business continuity and legal compliance with adequate performance, cost, and reliability.

Cross-references

- ▶ [Backup and Restore](#)
- ▶ [Checksum and Cyclic Redundancy Check Mechanism](#)
- ▶ [Continuous Data Protection](#)
- ▶ [Disaster Recovery](#)
- ▶ [Logical Volume Manager](#)
- ▶ [Point-in-Time Copy](#)
- ▶ [Redundant Arrays of Independent Disks](#)
- ▶ [Replication](#)
- ▶ [Write Once Read Many](#)

Recommended Reading

1. ANSI. NFPA1600 Standard on Disaster/Emergency Management and Business Continuity Programs.
2. BSI. BS25999; Business Continuity Management.

3. Houghton A. Error Coding for Engineers. Kluwer Academic Publications, Hingham, MA, 2001.
4. Keeton K., Santos C., Beyer D., Chase J., and Wilkes J. Designing for disasters. In Proc. 3rd USENIX Conf. on File and Storage Technologies, 2004.
5. Uchi N.K. System for recovering data stored in failed memory unit. US Patent 4,092,732, 1978.
6. Patterson D., Gibson G., and Katz R. A case for redundant arrays of inexpensive disks (RAID). In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988.
7. Sweeney P. Error Control Coding From Theory to Practice. Wiley, New York, 2002.
8. <http://www.sec.gov/>

Storage Protocols

KALADHAR VORUGANTI

Network Appliance, Sunnyvale, CA, USA

Synonyms

[iSCSI](#); [FCP](#); [Parallel SCSI](#); [SAS](#); [ATA](#); [SATA](#); [NFS](#); [CIFS](#); [Samba](#)

Definition

The emergence of networked storage has allowed organizations to de-couple their server and storage purchasing decisions. Thus, multiple application/database servers can share storage on the same network attached storage (NAS) server, or on a block storage controller via a storage area network protocol (SAN). This is unlike direct attached storage systems (DAS) where disks are connected to an application to a database server. In DAS environments, even if one only needs to add more storage, one has to also add more servers. Various types of storage networking protocols have emerged to support both NAS and SAN systems. Currently, the same storage boxes can support both distributed file (NAS) protocols such as NFS and CIFS, and block storage (SAN) protocols such as iSCSI and FCP (fiber channel). Most of the direct attached storage systems are moving from supporting parallel SCSI protocol to supporting serial ATA (SATA) or serial SCSI (SAS) protocols. The focus of this section is to set the context with respect to when they are used, and to compare the advantages and disadvantages of these different protocols.

Historical Background

Historically the fields of storage and networking were two separate fields. With the advent of network

attached storage the fields of storage and networking have now converged. Many of the existing networking protocols are now also used to carry storage payload from storage devices to the application servers.

Foundations

- *SAN Protocol Analysis:* SAN protocols transfer SCSI commands and data over a transport protocol. SAN protocol is between SCSI initiators and SCSI targets. The initiators typically reside on the host (application server), and the targets reside on a storage controller box. The SCSI initiator can be software based or it can reside in a host bus adapter card. In the SAN protocol space the comparison is primarily between Fiber Channel and iSCSI. Previously, IBM's SSA protocol competed with Fiber Channel, but has been discontinued as it was not universally adopted by all the storage controller vendors.
- *iSCSI:* iSCSI protocol transfers SCSI blocks and commands over TCP/IP networks. iSCSI protocol allows organizations to leverage their IP networks to transfer block storage data. Until the emergence of gigabit Ethernet networks, transfer of block storage data over Ethernet/TCP/IP stack proved to be not viable due to performance problems. Hardware TCP/IP off-load cards as well as multi-core CPU servers have alleviated the TCP/IP processing overhead. Software TCP/IP optimizations such as interrupt coalescing, and zero-copy optimizations have also reduced the CPU processing overhead. iSCSI based SANs have the following benefits over Fiber Channel based SANs:
 - *Interoperability:* Until recently, Fiber Channel devices from different vendors did not always interoperate with each other. Device interoperability certification is a much more elaborate and expensive process in the fiber channel world in comparison to the IP world.
 - *Distance:* IP networks have been designed to operate across large geographic distances. Fiber Channel networks have distance limitations and one typically needs channel extenders to extend their range.
 - *Cost:* An organization can leverage their IP network management expertise and devices for also transferring their storage traffic.

- *Security:* The security protocols for IP networks have been well developed. Since Fiber Channel networks have been typically used behind fire-walls, the security aspects are being still developed.
- *FCP (Fiber Channel):* Sending SCSI block commands and data over the Fiber Channel protocol is known as FCP. Fiber Channel protocol stack consists of physical, data link, network, and transport layer protocol specification. Fiber Channel protocol has the following benefits in comparison to TCP/IP stack:
 - *Performance:* Fiber Channel provides better performance than TCP/IP stack due to the following reasons:
 - *Hardware Offload:* The performance of first generation iSCSI HBAs was inferior to Fiber Channel HBAs. However, performance gap is being reduced in the newer generation iSCSI offload cards. The iSER and iWARP standards are trying to commoditize the RDMA standard for IP networks in order to bring down the cost of iSCSI offload cards. That is, these cards will be useful for additional protocols (not just iSCSI).
 - *Reservation Based Protocol:* Fiber Channel is a reservation based protocol instead of a retry based protocol. Thus, frames are not lost in Fiber Channel. The slow-start congestion control mechanism in TCP/IP is ill-suited for gigabit speed networks.

Currently, Fiber Channel protocol is the most commonly used SAN protocol in data centers.

- *Parallel versus Serial Protocols Analysis:* Parallel SCSI and ATA are two parallel protocols who have distance and device connectivity limitations. It is important to distinguish between the parallel SCSI transport protocol, and the SCSI block protocol which transfers data on top of the transport protocol. The physical wires for these parallel protocols are also quite wide and they make wiring a cumbersome process. Serial ATA (SATA), and Serial SCSI (SAS) have overcome the stringent distance and connectivity limitations of parallel SCSI and ATA transport mechanisms.

- *SCSI versus ATA Command Set Analysis:* The SCSI block protocol is transported over Fiber Channel (FCP), TCP/IP (iSCSI), and parallel SCSI, and serial SCSI (SAS) transport layers. The IDE/ATA protocol is transported over ATA and Serial ATA (SATA) transport mechanisms. It is important not to mix the block protocol with the underlying transport protocol. SCSI block and ATA block protocols have the following key differences:
 - *Command Queuing at the Device:* Previously SCSI protocol allowed for the queuing of multiple commands at the SCSI device, whereas, ATA protocol did not have this functionality. Recently, tagged command queuing functionality has been added to ATA protocol. Tagged command queuing also allows the device to optimize the order in which the queued commands are executed.
 - *Number of Connected Devices:* ATA protocol supported fewer devices per channel than the SCSI protocol. However, this limitation has been reduced in the serial ATA protocol with the emergence of port multipliers. Thus, 15 devices can be simultaneously supported using SATA.
 - *Checksums:* ATA protocol did not initially contain support for checksums on command and data. Later versions of the protocol have added checksum support for data.
 - *Hot-Plug of Devices:* ATA does not provide support for hot replacement of devices. Serial ATA has rectified this deficiency.
 - *Bus Mastering:* Previous versions ATA protocol did not allow to DMA data directly from the device into memory. However, recent versions of the protocol have overcome this limitation.
- *Block versus File-Based NAS Protocols:* NAS protocols provide a file level abstraction to the client applications, whereas SAN protocols provide a block level abstraction. NAS protocols allow for multiple hosts to share a file system namespace, whereas, SAN protocols allow multiple hosts to share a block level (SCSI device and LUN level) namespace. Previously, Fiber Channel based SANs provided better performance than IP based NAS protocols due to the implementation differences between the IP and Fiber Channel transport protocol stacks. However, this difference is disappearing due to the arrival of TCP/IP stack offload cards. In the past, NAS systems were not as scalable as SAN systems due to the absence of clustered NAS solutions but with the advent of clustered NAS solutions, this limitation has been also overcome.
- *NAS Protocol Analysis:* NFS and CIFS are the two primary NAS protocols. NAS protocols are between NAS clients that reside on the application/database server, and a NAS server that typically is a separate box that contains a file system and manages disks. NAS clients provide a file system interface to the host applications. NFS is used primarily in the Unix/Linux environment and it is an IETF standard, whereas, CIFS is used primarily in the Windows OS environment and its management/administrative APIs are proprietary (controlled by Microsoft). Samba is a popular protocol bundle that implements many different CIFS related protocols. A Samba server can act as an open-sourced CIFS server on Unix systems that makes Unix directories appear as Windows folders to Windows clients. With the emergence of NFSv4, many of the NFS deficiencies with respect to recovery management, caching, and security have been overcome and makes NFS a competitive protocol. NFSv4 has consolidated numerous protocols such as nfs, nlm, mountd, and nsm. NFSv4 is a stateful protocol, and it introduces the concept of delegation to allow for aggressive data caching at the clients [2].

Key Applications

SANs have been typically used by applications that want block level access to storage and are performance conscious. NAS solutions have been typically used by users who are more cost conscious and want to leverage their existing IP infra-structure and IP network management experience. In SANs, Fiber Channel and iSCSI protocols are typically used to connect application servers to the storage controller boxes. Fiber Channel, SATA and SAS protocols are typically used to connect the storage controller processor/cache complex to its backend arrays/disks. In DAS environments, the server is connected to its backend storage using Parallel SCSI, ATA, SATA or SAS. In NAS environments, NFS/CIFS are used to connect the application servers to the NAS server, and SATA, Fiber Channel or SAS is used to connect the NAS processor complex to the back-end disk arrays.

Cross-references

- ▶ Direct Attached Storage
- ▶ Network Attached Storage
- ▶ SAN File System
- ▶ Storage Area Network

Recommended Reading

1. Kaladhar V. and Prasenjit S. An analysis of three gigabit networking protocols for storage area networks. In Proc. 20th IEEE Int. Performance, Computing and Communications Conf., 2001.
2. Peter R., Li Y., Pawan G., Prasenjit S., and Prashant S. A performance comparison of NFS and iSCSI for IP-networked storage. In Proc. 3rd USENIX Conf. on File and Storage Technologies, 2004.

- Discovery and investigation of the utilization of all storage resources in a SAN, plus related integrated monitoring and integrated management operations.
- Analysis and reporting of storage resource utilization
- Analysis and estimation of movements in resource utilization, issuing alerts and/or execution of scripts for automation

Monitoring and indicating storage resource utilization from the business application viewpoint is an important requirement for storage resource management. To meet this requirement, storage resource management and storage configuration management, including dependency mapping between applications and storage devices, are often provided in combination.

Cross-references

- ▶ ILM
- ▶ LUN
- ▶ SAN
- ▶ Storage Consolidation
- ▶ Storage Management
- ▶ Storage Network Architectures
- ▶ Storage Virtualization

Recommended Reading

1. Storage Network Industry Association. Storage Network Industry Association tutorials, 2007. Available at: <http://www.snia.org/education/tutorials/>

Storage Resource Management

HIROSHI YOSHIDA

Fujitsu Limited, Yokohama, Japan

Synonyms

SRM

Definition

The management of physical and logical storage resources, including storage elements, storage devices, appliances, virtual devices, disk volume and file resources. In most cases, storage resource management is achieved by software tools which indicate and manage the storage resource utilization in a storage networking environment.

Key Points

When multiple storage devices are connected to a SAN and shared by multiple servers, the space in each device cannot be used uniformly, e.g., one device may be almost full while another is underutilized. If the storage administrator is unaware of the correct space utilization status, inefficient resource utilization of the whole SAN environment can occur. This may cause unnecessary addition of new storage resources, or additional resources are not installed in time causing business applications to stop due to lack of required storage space.

To indicate and manage storage resource utilization, storage resource management software tools provide the following functions:

Storage Security

KALADHAR VORUGANTI

Network Appliance, Sunnyvale, CA, USA

Synonyms

CAS; Authentication; Access control; WORM; Compliance; On-wire security; On-Disk security; Data corruption; Zoning; LUN masking; Port binding; Provenance; Watermarking

Definition

The definition of storage security has many facets, and some of the key requirements are:

Storage (and the appropriate manipulation capabilities) should only be accessible and visible to users with the appropriate permissions

Users should be notified if their data has been tampered with or altered either intentionally or unintentionally.

Malicious users should not be allowed to access or tamper with other people's storage. If possible, the system should be able to catch malicious users.

User's data should be physically deleted at the appropriate time. That is, it should not be present past its intended life-time.

Tracking unauthorized copying or access control delegations is also an important security concern.

Historical Background

Security has been a key computer science topic for many decades. Initially, people focused on authentication, encryption and access control for standalone systems. As distributed computing became popular, people started focusing on security within the context of network protocols. Now, with the emergence of utility computing, grid computing and cloud computing, storage is being accessed remotely. Thus, security issues pertaining to storage environment is now gaining importance. Many of the known security algorithms and techniques are now being re-used within the context of storage systems.

Foundations

The following different security mechanisms together provide the appropriate desired security functionality:

Storage Access Authentication: In order to prevent unauthorized users from accessing a storage device, there is usually an authentication mechanism employed (e.g., Kerberos, CHAP) to prevent unauthorized users from accessing the storage device. If the storage device is being accessed in a client-server manner, where the client resides on a database or application server, the client-server protocol needs to employ an authentication mechanism (e.g., Kerberos, or CHAP). In cases where both the host servers and the storage devices reside behind the firewall, client authentication is typically not performed.

On-Wire Security: When information flows between clients and storage servers outside the fire-wall protected domains, information is typically encrypted and sent over the wire to prevent eavesdropping. Data are also protected by a hash key to detect data tampering. IPSec security infra-structure provides authentication,

data encryption and tampering detection support at the IP layer for internet environments. Data are typically not encrypted on the wire in intranet environments behind the firewall.

On-Disk Encryption: Data stored on persistent storage (disks, tapes etc) is typically not encrypted because disks usually reside in secure places. However, with the emergence of storage service providers, an organization's data are managed by others. Thus, many organizations prefer to encrypt their data before storing it at a remote location. It is very important to store and remember the keys that have been used to encrypt the data because loss of the key is as good as the loss of the data. Organizations also periodically re-encrypt data to prevent the use of brute-force approaches to break the keys being used to encrypt the data. AES, DES and 3DES are some of the encryption algorithms that are being currently used.

Access-Control: Access control mechanisms have been in place and are used extensively in operating systems to control access to various resource types. Access control typically amounts to controlling read/write access for individual users, groups and everyone to files, directories of files, volumes, and storage devices. Access control mechanisms also deal with delegation of control by secondary data owners to third parties who want to access the data.

Compliance: Organizations are being expected to adhere to many different types of government regulations such as:

- There should be only a single copy of a particular type of data.
- Data should be physically erased from the storage medium after so many years. That is, data cannot be logically deleted, and one should not be able to reconstruct the data from the physically deleted copy.
- Organization should be able to provide a guarantee that data has not been altered since its creation. Many organizations employ WORM (write-once, read-many) type of storage medium such as optical storage or WORM tapes or CAS systems to attain this capability.
- During audits data belonging to a certain topic should be available within a short specified period of time (for example, data should be available within 48 h). In some cases this precludes the storing of data on tape.

Data Corruption: Data can be corrupted either unintentionally due to device failure or malfunction, or it could be corrupted intentionally by a malicious user or a virus. In either case, data corruption has to be detected and subsequently corrected. Data corruption can be detected at the time of data creation, or it can be checked for by a periodic checking process such as virus scans or disk scrubbing. Data corruption can be dealt with in the following ways:

- **Detection:** Data corruption can be detected (not corrected) using a hash function such as MD5 or SHA-1, checksum or CRC.
- **Detection and Correction:** Data corruption can be detected and corrected using error correction codes such as Reed-Solomon.
- **Correction:** Data corruption can be only corrected using erasure correcting codes such as a variant of Reed-Solomon codes.

Zoning: In storage area networks, zoning is a technique that is used to control which host port can have access to which storage device port. Zoning can be controlled at the switch firmware level (called hard zoning), by which one controls which switch input port can see which switch output port. Zoning can also be controlled at the Fiber Channel name service level (soft zoning), by which one controls which host server port can see which storage device port.

Port Binding: Port binding is the process by which one controls which port can be connected to a particular switch port. That is, one explicitly specifies the address of the host or storage device port that can exclusively transfer data to a particular switch port.

LUN Masking: The process of determining which host port can access which volume on the target via which target port is called as LUN masking. Thus, using this process one can control the access of storage volumes by the different hosts.

Provenance and Watermarking: With the emergence of peer to peer networks, data are getting copied at a very rapid rate often without the proper permission. In many cases, parts of the original document get copied and combined with new content. Thus, it is necessary to keep track of the trail of how data has been copied and modified from its source to its current location. In addition to keeping track of the original author and source of the data, it is also necessary to detect pirated copies of data. New digital document water-marking

techniques are being developed to add hidden copyright notices to documents.

CAS: Content addressable storage (CAS) systems provide a new type of data access mechanism. In CAS systems one generates a hash value out of the data content. This hash value is subsequently used to uniquely access the data. In CAS systems, one typically does not overwrite existing data but instead a new copy of the data gets created when updates are made to existing data copies. CAS systems have been used to provide WORM media capabilities for disk based systems.

Key Applications

Until recently, storage systems have been based behind company fire-walls. Thus, authentication and encryption have not been a major issue. Access control and data corruption (due to viruses or device failures) have been the major forms of security/integrity checking processes. With the emergence of government compliance regulations, data compliance has become a very major issue for most organizations. With the emergence of third party archival or storage service provider paradigms, more importance is being given to storage authentication and encryption mechanisms.

Cross-references

- ▶ [Access Control Administration Policies](#)
- ▶ [Access Control Policy Languages](#)
- ▶ [Asymmetric Encryption](#)
- ▶ [Authentication](#)
- ▶ [Data Encryption](#)
- ▶ [Database Security](#)
- ▶ [Discretionary Access Control](#)
- ▶ [Homomorphic Encryption](#)
- ▶ [Mandatory Access Control](#)
- ▶ [Message Authentication Codes](#)
- ▶ [Network Attached Secure Device](#)
- ▶ [Security Datawarehouses](#)
- ▶ [Security Services](#)
- ▶ [Symmetric Encryption](#)
- ▶ [Temporal Access Control](#)
- ▶ [XML Security](#)

S

Recommended Reading

1. Riedel E., Kallahalla M., and Swaminathan R. A framework for evaluating storage system security. In Proc. 1st USENIX Conf. on File and Storage Technologies, 2002.

Storage Servers

- ▶ Storage Devices

Storage Systems

- ▶ Performance Analysis of Transaction Processing Systems

Storage Virtualization

HIROSHI YOSHIDA

Fujitsu Limited, Yokohama, Japan

Definition

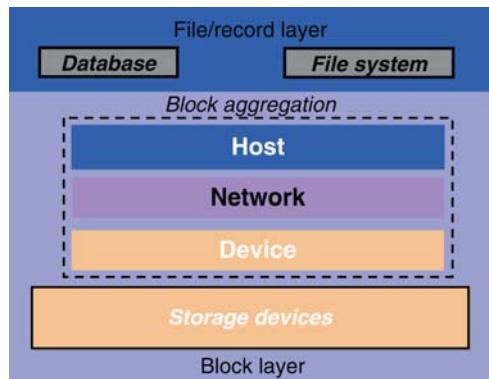
Storage virtualization is technology to build logical storage using physical storage devices. More precisely, SNIA defines storage virtualization as follows:

1. The act of abstracting, hiding, or isolating the internal function of a storage (sub) system or service from applications, compute servers or general network resources for the purpose of enabling application and network independent management of storage or data.
2. The application of virtualization to storage services or devices for the purpose of aggregating, hiding complexity or adding new capabilities to lower level storage resources.
3. Storage can be virtualized simultaneously in multiple layers of a system, for instance to create hierarchical storage manager like systems.

Key Points

Storage virtualization is used to provide an aggregation of data blocks (a logical volume) to applications running on servers. This aggregation of blocks may reside on a single storage device, may be spanned across multiple devices when it is too large to fit on a single device, may be mirrored to multiple storage devices to increase availability, or may be striped across multiple storage devices to enable parallel accesses.

Storage virtualization (block aggregation) can be implemented in various layers between the host server and the storage devices, i.e., in the host layer



Storage Virtualization. Figure 1. Block aggregation in shared storage model by SNIA.

(software-based logical volume manager), in the network layer (switch-based virtualization appliance), and in the device layer (disk array). [Figure 1](#) shows where block aggregation is implemented in the shared storage model defined by Storage Networking Association.

Storage virtualization technologies are now being extended to provide more capabilities. For example, multiple storage devices which reside in different locations can be virtualized as a single logical storage device with disaster recovery capability. Another example is the virtualization of inexpensive storage devices with high capacity together with expensive devices with high performance to form a single logical device where migration between the physical devices is done automatically. This provides a cost-effective storage system suitable for hierarchical storage management or information lifecycle management (ILM).

Cross-references

- ▶ ILM
- ▶ SAN
- ▶ SRM
- ▶ Storage Consolidation
- ▶ Storage Network Architectures
- ▶ Storage Networking Industry Association

Recommended Reading

1. Storage Network Industry Association. Storage Virtualization: the SNIA technical tutorials, 2007. Available at: http://www.snia.org/education/storage_networking_primer/stor_virt/
2. Storage Network Industry Association. The SNIA shared storage model, 2007. Available at: http://www.snia.org/education/storage_networking_primer/shared_storage_model/

Stored Procedure

TORE RISCH
Uppsala University, Uppsala, Sweden

Definition

Modern relational query languages such as SQL provide general programming language capabilities in addition to the statements for searching and updating the database. A stored procedure is a user program written in a query language running inside the database server. Stored procedures often include side effects that update the database. This makes it possible to define general programs using the query language. These programs are called stored procedures and are executed inside the database server.

Key Points

In SQL, the user defines stored procedures as a schema manipulation statement using as CREATE PROCEDURE statement. A so defined procedure is immediately shipped to the database server where it is compiled and stored. Stored procedures can have side effects that change the state of the database. In order to prohibit searches that change the state of the database, in SQL stored procedures cannot be called from within queries, but only from applications or general SQL interfaces. SQL provides a special EXEC statement passing to the procedure when it is called.

The advantage with stored procedures are:

- Communication time is saved in case the communication with the application program is slow or unreliable.
- Common database centered updates and computations belong naturally to the database.

A disadvantage with stored procedures is that different vendors often provide different stored procedure languages. The SQL-PSM (Persistent Stored Modules) standard defines stored procedures in SQL.

Related to stored procedures (and part of SQL-PSM) are user defined functions (UDFs), which are user defined functions to perform common side-effect free computations. UDFs are allowed as expressions in queries.

Cross-references

- ▶ [Query Language](#)

S-Transactions

- ▶ [Flex Transactions](#)

Stream Data Analysis

- ▶ [Stream Mining](#)

Stream Mining

JIAWEI HAN, BOLIN DING
University of Illinois at Urbana-Champaign,
Champaign, IL, USA

Synonyms

[Stream data analysis](#)

Definition

Stream mining is the process of discovering knowledge or patterns from continuous data streams. Unlike traditional data sets, *data streams* consist of sequences of data instances that flow in and out of a system *continuously* and with varying update rates. They are *temporally ordered, fast changing, massive, and potentially infinite*. Examples of data streams include data generated by communication networks, Internet traffic, online stock or business transactions, electric power grids, industry production processes, scientific and engineering experiments, and video, audio or remote sensing data from cameras, satellites, and sensor networks. Since it is usually impossible to store an entire data stream, or to scan through it multiple times due to its tremendous volume, most stream mining algorithms are confined to reading only once or a small number of times using limited computing and storage capabilities. Moreover, much of stream data resides at a rather low level of abstraction, whereas analysts are often interested in relatively high-level dynamic changes, such as trends and deviations. Therefore, it is essential to develop online, multilevel, multi-dimensional stream mining methods. Stream mining can be considered a subfield of data mining, machine learning, and knowledge discovery.

Historical Background

There are extensive studies on stream data management and the processing of continuous queries in stream data [4]. Different from stream query processing, stream mining extracts patterns and knowledge from online stream data. It covers the topics of mining multidimensional stream statistics, frequent patterns, classification models, clusters, and outliers in online data streams. Substantial research on stream mining has appeared since only 2000, almost at the same time as the research on stream data management. However, lots of results have been generated in this line of research.

Foundations

Stream mining problems are challenging because of the following two reasons. (i) Stream data are massive, arriving with high speed, and updated frequently, so that one can neither store all the data nor scan the data repeatedly, and in the meantime, the response time is usually required to be short in applications; therefore, stream synopsis construction is popularly used to maintain a summary of stream data online using limited space without losing too much information. (ii) Stream data often evolves considerably over time, and for classification or clustering, one should often use biased sampling of the stream data to emphasize more recent behavior of the stream. In general, stream mining can be partitioned into four themes: (i) online computing multidimensional stream statistics, (ii) mining frequent items and itemsets over stream data, (iii) stream data classification, and (iv) clustering data streams.

For online computing multidimensional stream statistics, a multidimensional stream cube model was proposed by Chen et al. [6] in their study of multidimensional regression analysis of time-series data streams. A stream cube can be efficiently constructed based on (i) a tilted timeframe, where the finer granularity is used for more recent time, and the coarser granularity for more distant time, (ii) a minimal interest layer to register the minimal layer of the cube that is still of user's interest, and an observation layer for the cuboid that a user usually watches for trends or anomaly, and (iii) partial materialization that materializes the cuboids only along the popular drilling path, serving as a tradeoff between the storage space and online response time. Statstream, a statistical

method for the monitoring of thousands of data streams in real time, was developed by Shasha and Zhu [15].

For mining frequent items and itemsets over stream data, one important issue is how to return approximate frequency counts for items or itemsets with limited buffer size for infinite data streams. Manku and Motwani [12] proposed Sticky Sampling algorithm and Lossy Counting algorithm for computing approximate frequency counts of items over data streams, and developed a Lossy Counting based algorithm for computing frequency counts of itemsets with the focus on system-level issues and implementation artifices. Another important issue is how to track frequent items dynamically. "Dynamically" means data streams consist of both "insertion" operations and "deletion" operations of items (imagine cars entering and exiting the parking lot). Cormode and Muthukrishnan [7] proposed algorithms for this problem using group testing and randomization techniques. Keeping track of frequent of items in such data streams arises in applications of both traditional databases and other domains, like telecommunication networks.

For stream data classification, the goal is to predict the class label or the value of new instances in the data stream, given some knowledge about the class membership or values of previous instances in the data stream. Since the distribution underlying the instances or the rules underlying their labeling may change over time, the class label or the target value to be predicted may change over time as well (stream data is evolving). This problem is referred to as *concept drift*. A major challenge in stream classification is how to construct highly accurate models with the existence of concept drift in stream data. Hulten et al. [10] developed an algorithm CVFDT, by integrating concept drift in time-changing data streams and a statistical measure, Hoeffding bound. Wang et al. [16] proposed an ensemble classifier to mine concept-drifting data streams. Aggarwal et al. [3] developed a k -nearest neighbor-based method for classify evolving data streams. Gao et al. [8] handled skewed distributions and proposed an ensemble-based framework that under-samples the overwhelming negative data, and repeatedly samples the scarce positive data for model construction with concept-drifting data streams.

For clustering data streams, in some applications, simply assume that the clusters are to be computed

over the entire data stream, and view the stream clustering problem as a variant of single-scan clustering algorithms. For example, Guha et al. [9] gave space-efficient constant-factor approximation algorithms for the k-median problem in stream data using divide-and-conquer and randomization techniques; O'Callaghan et al. [14] proposed a *k*-median based stream clustering method by incrementally updating *k*-median centers. However, it is important to consider stream evolution over time besides the single-scan constraint and resource limitation. When stream data is evolving, the underlying clusters may also change considerably over time. If the entire data stream is used for clustering, the result is likely to be inaccurate. What's more, at one moment, users may wish to examine clusters occurring in different time periods (e.g., last week/month/year). Aggarwal et al. [2] proposed a *CluStream* framework for clustering evolving data streams by introducing a tilted time-frame, an online microclustering maintenance, and an offline query-based macro-clustering mechanism, to achieve efficiency and high clustering quality and to provide the flexibility to compute clusters over user-defined time periods in an interactive fashion.

Aggarwal [1] provides a comprehensive survey on stream data processing and stream mining with a collection of chapters on difference issues on stream mining.

Key Applications

There are broad applications of stream mining, of which only a few examples are illustrated.

Mining anomaly in network streams. Stream mining has been popularly used for mining anomaly in computer network or other stream data. For example, MAIDS (Mining Alarming Incidents in Data Streams) [5] is a system that explores tilted timeframe and stream cube for mining computer network anomaly.

Computing statistical measures over time series data streams. Another popular application is to compute statistical measures over time series streams, such as StatStream [15].

Integration of mobile computing with stream mining. Kargupta et al. [11] have developed VEDAS (Vehicle Data Stream Mining System) that allows continuous monitoring and pattern extraction from data streams generated onboard a moving vehicle.

Future Directions

There are many challenging issues to be researched further. Only a few are listed below.

Mining sophisticated patterns in data streams. Due to the single-scan constraint and resource limitation, the current stream pattern mining methods are confined to simple patterns, such as single items or some limited itemsets. Tasks for mining sequential patterns [13] and structured patterns are challenging but interesting for research.

Mining sophisticated data sets for advanced applications. Text (e.g., document) streams and video streams are important real-life stream mining applications. However, it is challenging to mine such streams because it requires both sophisticated text/video analysis and real-time resource constraints. Other advanced stream mining applications include spatial data streams, financial transaction data streams, and so on.

Experimental Results

There are many experimental results reported in numerous conference proceedings and journals.

Data Sets

UCI Machine Learning Repository: <http://archive.ics.uci.edu/ml/datasets.html>

URL to Code

RapidMiner (previously called YALE (Yet Another Learning Environment)), at <http://rapid-i.com>, is a free open-source software for knowledge discovery, data mining, and machine learning. It also features data stream mining, learning time-varying concepts, and tracking drifting concept.

MassDAL (Massive Data Analysis Lab) Public Code Bank, at <http://www.cs.rutgers.edu/muthu/massdal-code-index.html>, is a library of routines in C and Java for stream data and other massive data set analysis, including implementations of some published algorithms for finding frequent items over stream data.

Cross-references

- [Association Rule Mining on Streams](#)
- [Classification in Streams](#)
- [Clustering on Streams](#)
- [Data Mining](#)
- [Event Stream](#)
- [Frequent Items on Streams](#)

- ▶ Geometric Stream Mining
- ▶ Stream Similarity Mining
- ▶ Wavelets on Streams

Recommended Reading

1. Aggarwal C.C. Data Streams: Models and Algorithms. Kluwer Academic, 2006.
2. Aggarwal C.C., Han J., Wang J., and Yu P.S. A framework for clustering evolving data streams. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 81–92.
3. Aggarwal C.C., Han J., Wang J., and Yu P.S. On demand classification of data streams. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2004, pp. 503–508.
4. Babcock B., Babu S., Datar M., Motwani R., and Widom J. Models and issues in data stream systems. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 1–16.
5. Cai Y.D., Clutter D., Pape G., Han J., Welge M., and Auvin L. MAIDS: Mining alarming incidents from data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 919–920.
6. Chen Y., Dong G., Han J., Wah B.W., and Wang J. Multi-dimensional regression analysis of time-series data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 323–334.
7. Cormode G. and Muthukrishnan S. What's hot and what's not : tracking most frequent items dynamically. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003, pp. 296–306.
8. Gao J., Fan W., Han J., and Yu P.S. A general framework for mining concept-drifting data streams with skewed distributions. In Proc. SIAM International Conference on Data Mining, 2007.
9. Guha S., Mishra N., Motwani R., and O'Callaghan L. Clustering data streams. In Proc. 41st Annual Symp. on Foundations of Computer Science, 2000, pp. 359–366.
10. Hulten G., Spencer L., and Domingos P. Mining time-changing data streams. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001.
11. Kargupta H., Bhargava B., Liu K., Powers M., Blair P., Bushra S., Dull J., Sarkar K., Klein M., Vasa M., and Handy D. VEDAS: A mobile and distributed data stream mining system for real-time vehicle monitoring. In Proc. SIAM International Conference on Data Mining, 2004.
12. Manku G. and Motwani R. Approximate frequency counts over data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 346–357.
13. Mendes L., Ding B., and Han J. Stream sequential pattern mining with precise error bounds. In Proc. 2008 IEEE Int. Conf. on Data Mining, 2008.
14. O'Callaghan L., Meyerson A., Motwani R., Mishra N., and Guha S. Streaming-data algorithms for high-quality clustering. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 685–696.
15. Shasha D. and Zhu Y. High Performance Discovery In Time Series : Techniques and Case Studies. Springer, 2004.
16. Wang H., Fan W., Yu P.S., and Han J. Mining concept-drifting data streams using ensemble classifiers. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 226–235.

Stream Models

LUKASZ GOLAB

AT&T Labs-Research, Florham Park, NJ, USA

Definition

Conceptually, a *data stream* is a sequence of data items that collectively describe one or more underlying signals. For instance, a network traffic stream describes the type and volume of data transmitted among nodes in the network; one possible signal is a mapping between pairs of source and destination IP addresses to the number of bytes transmitted from the given source to the given destination. A stream model explains how to reconstruct the underlying signals from individual stream items. Thus, understanding the model is a prerequisite for *stream processing* and *stream mining*. In particular, the computational complexity of a data stream problem often depends on the complexity of the model that describes the input.

Historical Background

The stream models discussed in this article were introduced in [3] and extended in [7,8]. In addition to modeling a stream with respect to its underlying signal(s), there exist the following two related concepts. First, the stream computational model asserts that a stream algorithm must run in limited space and time, and can only make a small number of passes (often only one pass) over the data [8,6]. Furthermore, one can also model various statistical properties of a data stream, such as changes in the frequency distribution of the underlying signals or inter-arrival times of stream items (see, e.g., [9]).

Foundations

Basic Model Definitions

Consider a *data stream* S composed of individual items s_1, s_2, \dots , ordered by arrival time. Let A be a signal described by S . Assume that A is a function from

a discrete and ordered domain to the range of reals; i.e., $A : [1\dots N] \rightarrow \mathbb{R}$. For instance, in the motivating example from above, the domain consists of IP address pairs. That is, $N = 2^{64}$ since an IP address is 32 bits long.

There are four models for representing A using individual stream items:

1. In the aggregate model, each stream item s_i corresponds to a range value for some domain value.
2. In the cash register model, each stream item s_i represents a domain value and a partial range value r_p such that $r_p \geq 0$. Reconstructing the signal A involves aggregating all the r_i values corresponding to each domain value.
3. The turnstile model generalizes the cash register model by allowing any r_i to be negative. Thus, reconstructing the signal A involves adding/subtracting the contributions of stream items having positive/negative range values.
4. In the reset model, each stream item s_i corresponds to a range value and is understood to replace all previously reported range values for the given domain value.

Each of the four models defined above has an ordered and an unordered version. In the ordered version, stream items arrive over time in increasing order of the domain values. In the unordered version, the ordering of the domain does not correspond to the arrival order of stream items.

In terms of complexity and expressive power, the turnstile model is the most general, followed by the cash register model and the aggregate model, respectively. As a result, designing stream algorithms for the turnstile model is the most challenging. For instance, while many types of sketches have provable time and accuracy bounds in the turnstile and cash register models, *stream sampling* algorithms are typically applicable only in the cash register model [8] (effectively, the turnstile model allows deletions via negative range values, therefore it may not be possible to maintain a fixed-size sample of the stream over time). Additionally, geometric problems over streams (e.g., estimating the diameter of a stream of points) are difficult to solve in the turnstile model since previously seen points may be deleted in the future [8]. The reset model is also quite general and some algorithms in this model are more complex than in the other three models [8]. See [3,7,8] for examples of

algorithms and complexity bounds for different stream models.

Examples and Extensions of Basic Models

Consider a network traffic stream S composed of IP packets. Each packet contains (among other things) the source IP address, destination IP address, and size. Define signal A_1 as a function from the source and destination address pairs to the total number of bytes exchanged by each pair (i.e., sums of sizes of all the packets sent between a given pair). Since many packets may be exchanged between two nodes and packets may arrive in random order, this example corresponds to the unordered cash register model.

For an example of an ordered cash register model, define S_2 to be the output stream of a pipelined query plan for the following query:

```
SELECT a1, a2, count(*)
FROM T
GROUP BY a1, a2
ORDER BY a1, a2
```

Define signal A_2 as a function from values of a_1 to the corresponding frequency counts (aggregated over all possible values of a_2). This example corresponds to the ordered cash register model – stream items arrive in the order of their domain values due to the ORDER BY clause, but must be aggregated on a_1 in order to reconstruct A_2 .

Now suppose that the output stream of the above query is not ordered by a_1 , but has the property that all the groups having the same value of a_1 are streamed out contiguously. This conforms to the contiguous unordered cash register model. Note that the ordered cash register model is always contiguous.

Next, suppose that stream S_3 is a pre-processed version of S , where each item s_i is a triple of the form: (source IP address, destination IP address, event), where the event field denotes the start or end of a connection between two nodes. Define signal A_3 as a function from the source and destination address pairs to the total number of open connections between each pair that have not yet ended. This corresponds to the unordered turnstile model since a stream item carrying an end-of-connection event decrements the total count of open connections for the given pair of nodes; furthermore nodes may open and close connections in arbitrary order.

In the above examples, the range of the signal corresponds to non-negative integers. A turnstile

model whose signal has a non-negative range is said to be a strict turnstile model. For an example of a non-strict turnstile model, consider tracking the difference between the number of connections originating from two different IP addresses. Note that some sketch-based algorithms that work in the strict turnstile model do not apply in the non-strict version [8].

As in the cash register model, one may define a contiguous unordered (strict or non-strict) turnstile model; the ordered (strict or non-strict) turnstile model is always contiguous.

For an example of an aggregate model, suppose that stream S_4 is a pre-processed version of S , where each stream item denotes the total number of bytes exchanged between a given source-destination pair over a 5 min. *window*. Define signal A_4 as a function from the source and destination address pairs to the total number of bytes exchanged by each pair in the window. This gives rise to an ordered aggregate model if stream items are ordered by the source and destination addresses, and an unordered aggregate model otherwise. Note that S_4 may contain a concatenation of many instances of A_4 , each corresponding to range values calculated over a particular 5 min. *window*.

Alternatively, one can model S_4 as carrying a single signal over time, call it A_5 , whose domain is the same as that of A_4 , but whose range is total number of bytes exchanged by each pair in the most recent 5 min. interval. This corresponds to the (unordered and non-contiguous) reset model because new items arriving on the stream (i.e., those corresponding to the most recent 5 min. window) replace old items having the same domain value.

Key Applications

The unordered cash register model is appropriate for applications where the incoming stream contains multiplexed data feeds from many sources (e.g., network traffic). However, the turnstile model must be used if the input (which may be a pre-processed version of a stream originally conforming to the cash register model) includes positive and negative range values. In particular, the turnstile model can represent a signal whose range values are computed over sliding *windows*. To see this, note that values that expire from the window may be modeled as new stream items with negative range values. Stream items whose

purpose is to invalidate previously arrived items are often referred to as negative tuples [1,2,5].

On the other hand, the aggregate model is appropriate for many types of time series data, e.g., aggregated network traffic data generated every 5 min. In some cases, the reset model may also be suitable. Moreover, the reset model is useful in applications that process locations of moving objects over time. In these applications, moving objects (e.g., a fleet of delivery trucks) periodically report their current positions [7].

Future Directions

The four stream model described in this article can express a wide range of application scenarios. However, new *streaming applications*, such as *publish-subscribe over streams*, *XML-stream processing*, signal-oriented stream processing [4], and analysis of the results of large-scale scientific experiments, may require new models.

Cross-references

- ▶ [Data Stream](#)
- ▶ [Stream Mining](#)
- ▶ [Stream Processing](#)

Recommended Reading

1. Arasu A., Babu S., and Widom J. The CQL continuous query language: semantic foundations and query execution. VLDB J., 15(2):121–142, 2006.
2. Ghanem T., Hammad M., Mokbel M., Aref W., and Elmagarmid A. Incremental evaluation of sliding-window queries over data streams. IEEE Trans. Knowl. and Data Eng., 19(1):57–72, 2007.
3. Gilbert A., Kotidis Y., Muthukrishnan S., and Strauss M. Surfing wavelets on streams: one-pass summaries for approximate aggregate queries. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 79–88.
4. Girod L., Mei Y., Newton R., Rost S., Thiagarajan A., Balakrishnan H. and Madden S. The case for a signal-oriented data stream management system. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 397–406.
5. Golab L. and Özsu M.T. Update-pattern aware modeling and processing of continuous queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 658–669.
6. Henzinger M., Raghavan P., and Rajagopalan S. Computing on data streams. DIMACS Ser. Discrete Math. Theor. Comput. Sci., 50:107–118, 1999.
7. Hoffmann M., Muthukrishnan S., and Raman R. Streaming algorithms for data in motion. ESCAPE. Springer, Berlin Hiedelberg New York, 2007, pp. 294–304.
8. Muthukrishnan S. Data streams: algorithms and applications. Found. Trends Theor. Comput. Sci., 1(2):1–67, 2005.
9. Paxson V. and Floyd S. Wide-area traffic: the failure of Poisson modeling. IEEE/ACM Trans. Netw., 3(3):226–244, 1995.

Stream Processing

MICHAEL STONEBRAKER^{1,2}

¹Massachusetts Institute of Technology, Cambridge, MA, USA

²StreamBase Systems, Inc., Lexington, MA, USA

Synonyms

Complex event processing (CEP); Event stream processing (ESP); Data stream processing

Definition

Stream processing refers to a class of software systems that deals with processing streams of high volume messages with very low latency. It is distinguished from business activity monitoring (BAM) or business process monitoring (BPM), in that the client of a stream processing application is often a program, rather than a human. Hence, the volume and latency requirements are often much more stringent.

Currently, stream processing is widely used in computing real-time analytics in e-trading, maintaining the state of massively multi-player Internet games, real-time risk analysis, network monitoring, and national security applications. In the future, the declining cost of sensor technology will create new markets for this technology including congestion-based tolling on freeways and prevention of lost children at amusement parks.

Key Points

There are three main technical approaches to stream processing at the present time.

- *Custom code.* Traditionally, stream processing applications have been hand-coded in a low-level programming language such as C or C++. The current trend is toward using one of the other two technologies to achieve lower development and maintenance cost.
- *Stream-oriented SQL.* Recent research activity has extended SQL with primitives for real-time operation. The main additions are the notion of real-time windows, over which SQL aggregates can be computed, facilities to perform pattern matching on sequences of messages, and primitives to deal with out-of-order data. There are now high-performance Stream-oriented SQL engines from several vendors (e.g., Coral8, <http://www.coral8.com>) and StreamBase, <http://www.streambase.com>).

com and StreamBase, <http://www.streambase.com>).

- *Rule engines.* The final approach is to utilize a high-performance implementation of a rule engine for stream processing. These systems are descendants of the rule engines found in expert systems in the 1980's and originally specified by the Artificial Intelligence community in pioneering work in the 1970s. Such systems contain rich pattern matching capabilities, but must be extended with aggregation and windowing constructs. Currently, there are a variety of commercial rule engines addressing the stream processing market.

In cases where real-time processing must be combined with access to historical data, Stream-oriented SQL enjoys a natural advantage. Both real-time and historical analysis can be done in a single paradigm (SQL), whereas a rule engine must switch paradigms to access historical data. On the other hand, where very sophisticated pattern matching is the main requirement, rule engines enjoy an advantage, due to their richer pattern matching capabilities.

In either case, achieving high performance and low latency requires a collection of implementation optimizations. Extensive compilation, often to machine code, will lower message processing overhead. In addition, some systems go to great lengths to remove scheduling overhead, by pre-computing what operation must be performed next, and then directly calling that operation from the current one, thereby removing both scheduling and message queuing overhead. In addition, implementations based on storing real-time state in a DBMS (either a conventional disk-based one or a main memory DBMS) are not likely to be successful, because of the inherent overhead of these class of products.

Over the course of the next decade, it is expected that stream processing products will enter the mainstream, thereby complementing other system software components such as application servers and DBMSs.

Cross-references

- ▶ [Data Stream](#)
- ▶ [Data Stream Management Architectures and Prototypes](#)
- ▶ [Streaming Applications](#)
- ▶ [Stream-Oriented Query Languages and Operators](#)

Recommended Reading

- Stonebraker M., Çetintemel U., and Zdonik S. The 8 requirements of real-time stream processing. ACM SIGMOD Rec., 34(4):420–447, 2005.

Stream Query Processing

► Window-based Query Processing

Stream Sampling

BIBUDH LAHIRI, SRIKANTA TIRTHAPURA
Iowa State University, Ames, IA, USA

Definition

Stream sampling is the process of collecting a representative sample of the elements of a data stream. The sample is usually much smaller than the entire stream, but can be designed to retain many important characteristics of the stream, and can be used to estimate many important aggregates on the stream. Unlike sampling from a stored data set, stream sampling must be performed online, when the data arrives. Any element that is not stored within the sample is lost forever, and cannot be retrieved. This article discusses various methods of sampling from a data stream and applications of these methods.

Historical Background

An early algorithm to maintain a random sample of a data stream is the *reservoir sampling* algorithm due to Vitter [15]. More recent random sampling based algorithms have been inspired by the work of Alon et al. [1]. Random sampling has for a long time been used to process data within stored databases – the reader is referred to [13] for a survey.

Foundations

A powerful sampling technique is *random sampling*, where random elements of the stream are selected into the sample. A random sample of a stream can be used in deriving *approximate* answers to aggregate queries such as *quantiles* [12] or *frequent elements* [11]. It can also be used in the estimation of the *selectivity of a query predicate*, which is defined as the fraction of the data items in the stream which satisfy

the given user predicate. The intuition behind the above applications of random sampling is as follows. Suppose R is a uniform random sample of a data stream S . For any $A \subset S$, the size of A , $|A|$ can be estimated as $|A \cap R| \cdot |S| / |R|$. The accuracy of this estimate depends on the value of $|A \cap R| / |R|$. There are tradeoffs between the quality of the answer returned, the confidence in the answer, and the space taken by the sample.

There are two basic ways of generating a random sample of any data set – sampling without replacement and sampling with replacement. Consider a data stream with N elements and a sample size n . In random sampling with replacement, each element of the sample is chosen at random from among all N elements of the data set. It is possible that the same element is chosen more than once into the sample (though this is unlikely if the sample is much smaller than the data). A random sample without replacement is a randomly chosen subset of n elements from among all $\binom{N}{n}$ subsets of size n , thus ensuring that a data element appears no more than once in the random sample.

Reservoir Sampling

This technique, due to Vitter [15], allows the maintenance of a random sample of the stream of a particular target size in an online fashion. Suppose the objective is to maintain a random sample of n elements without replacement, from a stream of N elements, where N is not known a priori. Let the stream elements be a_1, a_2, \dots, a_N .

A discussion of how to maintain a sample of a single element from the stream i.e., the case $n = 1$, can be a good point to begin. When a_1 arrives, it is always selected into the sample. For $i \geq 2$, element a_i is selected into the sample with probability $1/i$, i.e., a_2 is selected with a probability $1/2$, a_3 is selected with a probability $1/3$, and so on. Each time an element is selected into the sample, it replaces the existing element in the sample. It can be verified that the final element in the sample has an equal probability of being any of the N elements in the stream.

The above idea can be extended to maintain a random sample without replacement of size n as follows. The first n elements of the stream are (deterministically) included in the sample. For $t \geq n$, when a_{t+1} arrives, it is included in the sample with probability $n/(t+1)$. If an element is selected for inclusion in the sample, it replaces an element that is chosen uniformly

at random from the currently existing elements in the sample. It is easy to verify that the resulting sample is equally likely to be any of the $\binom{t+1}{n}$ subsets of size n of the set a_1, a_2, \dots, a_{t+1} . This algorithm is described in Fig. 1.

Note that if one wanted to sample n elements from a stream *with replacement*, this could be achieved by running n copies of the single element reservoir sampling algorithm. Further enhancements are possible to the algorithm in Fig. 1. In particular, instead of examining every element of the stream to see if it will be sampled, it is possible to directly generate the number of elements of the stream to be *skipped* before the next element that will be included in the sample. This can significantly reduce the number of stream elements to be examined by the sampling algorithm. For further details, the reader is referred to [15].

Sample and Count

This is a technique pioneered by Alon et al. [1], and is based on random sampling followed by counting. This technique has been applied in the estimation of frequency dependent statistics on a data stream in very small space. To see its use in the context of estimating the frequency moments of a data stream, consider a stream $S = a_1, a_2, \dots, a_N$, where each $a_i \in \{1, 2, \dots, m\}$. For $1 \leq j \leq m$, let f_j denote the number of occurrences of j in stream S . For integral $k \geq 0$, the k th frequency moment of S , denoted by F_k is defined as follows.

$$F_k = \sum_{j=1}^m f_j^k$$

For $k \neq 1$, computing F_k exactly on a large data stream S is provably expensive space-wise. There are lower bounds showing that such an exact computation of F_k , or even an accurate *deterministic* approximation

of F_k requires $\Omega(m)$ space, in the worst case. However, a randomized approximation to F_k (for $k \geq 2$) can be found as follows. First, choose a random element a_p from S (this can be done without a knowledge of N using the reservoir sampling technique). Then maintain the count $X = |\{q : q \geq p, a_q = a_p\}|$. In other words, count the number of re-occurrences of the element a_p in the portion of the stream that succeeds a_p (including a_p). Then, the random variable $Y = N[X^k - (X-1)^k]$ is an unbiased estimator of F_k , i.e., $E[Y] = F_k$. Further, it can also be shown that the variance of Y is small. For user defined parameters $0 < \varepsilon, \delta < 1$, this can be used to generate an estimator of F_k that is within a relative error of ε with probability more than $1 - \delta$ using small space. For exact space bounds, proofs and details, the reader is referred to [1].

The sample and count technique has also been used in accurate estimation of another frequency dependent aggregate, the *empirical entropy* of a data stream, in limited space. Consider a stream of integers, $S = a_1, a_2, \dots, a_N$, where each $a_i \in \{1, 2, \dots, m\}$. For $1 \leq j \leq m$, let f_j denote the number of occurrences of j in S . The empirical entropy of the stream is defined as $\sum_{j=1}^m - (f_j/N) \log (f_j/N)$. The entropy of a stream yields valuable information about the amount of “randomness” within the stream, and is useful in many contexts in network monitoring. Chakrabarti et al. [3] present an algorithm for estimating the entropy of a stream. Their algorithm uses the sample and count technique, and yields a provably accurate estimate of the entropy (a randomized approximate estimate) using nearly optimal space.

Distinct Sampling

In some cases, it may be necessary to compute aggregates over all the *distinct* elements in the stream.

Input: Stream a_1, a_2, \dots, a_N where N is not known in advance; Sample Size n
Output: A sample $S[1, \dots, n]$ of n elements chosen uniformly at random without replacement from the stream.

1. For i from 1 to n , $S[i] \leftarrow a_i$ // *The first n elements are included in the sample*
 2. $t \leftarrow n$ // *t is the number of records processed so far*
 3. While (there are more stream elements)
 - (a) $t \leftarrow (t + 1)$
 - (b) Let s be a random number in the set $\{1, 2, 3, \dots, t\}$
-

Stream Sampling. Figure 1. Reservoir sampling for sampling without replacement from a stream.

Consider a stream of tuples (i, v) where i is an item identifier and v is the value. In database query optimization, a question of interest is often just an estimate of the number of distinct values for an attribute in a relation. In network monitoring, an objective may be tracking all those sources that have contacted a large number of distinct destinations in the recent past. For computing such aggregates over all distinct identifiers, a uniform random sample will not be useful. For example, suppose a stream of 10^8 elements had 1,000 distinct identifiers, but every identifier appeared exactly once in the stream, except for the “dominant identifier”, which made up the remaining $10^8 - 999$ elements of the stream. Even if a fairly large random sample of this stream is collected, say a sample of 10^4 elements, the sample is likely to contain only the dominant identifier, and any estimate from this sample is likely to be extremely inaccurate. To derive an useful sample for estimating aggregates over distinct elements in a stream, Gibbons and Tirthapura [9] introduced a technique called *distinct sampling*.

In distinct sampling, the sampling is performed with the help of a randomly chosen *hash function*, rather than through independent random choices for each element. The hash function h is chosen before the stream elements are observed. Given a target sample size n , the algorithm in Fig. 2 maintains a random sample of all distinct elements of a stream S of size approximately n . For example, suppose the stream of 10^8 elements had 7,500 distinct identifiers. If the target sample size was 10^4 , then all distinct elements would be

included in the sample. If the target sample size was 10^3 , then each distinct element would be included in the sample with probability p , where p is reduced until the resulting sample fits within the target sample size.

Let the stream S be an integer sequence a_1, a_2, \dots, a_N . The algorithm maintains a sample D of distinct elements, and a sampling level ℓ . At level ℓ , each distinct element is selected into the sample with probability $1/2^\ell$. For simplicity, assume a random hash function h is available such that $h(x)$ is a random real number which is uniformly chosen from the range $(0, 1)$, and the outputs of the hash function on different inputs are mutually independent. The analysis of the algorithm using more practical hash functions with limited independence (and integral outputs) is presented in [9].

The above distinct sampling algorithm assumed that all elements in the stream have the same weight – in general, this may not be true. For example, a user might be interested in computing the mean of the values received over all distinct identifiers in the stream – in this case, different elements should be weighted differently. The above algorithm was extended to the more general weighted case by Pavan and Tirthapura [14], who designed a “range-sampling” algorithm that allowed the weighted sampling problem to be reduced to the unweighted case.

Time-Decayed Sampling

The reservoir sampling (distinct sampling) algorithm maintains a sample of all elements (distinct elements) from the start of time, and such a sample is useful in

Input: Stream $S = a_1, a_2, \dots, a_N$ where $a_i \in \mathcal{I}$ and N is not known in advance; Maximum sample size n ; Hash function $h : \mathcal{I} \rightarrow (0, 1)$.

Output: D , a random sample of the distinct elements of the stream, chosen without replacement; The size of D is guaranteed not to exceed n .

1. Initialization: $D \leftarrow \emptyset, \ell \leftarrow 0$
 2. While there are more elements in S
 - (a) Let a be the next element in S
 - (b) If $(h(a) < \frac{1}{2^\ell})$
 - i. If $(a \notin D)$ then Add a to D
 - ii. If $(|D| > n)$ then || *Overflow, discard approximately half the items in D by selecting every item currently in D with a probability $1/2$.*
 - A. $\ell \leftarrow \ell + 1$
 - B. For every $a \in D$, if $h(a) \geq \frac{1}{2^\ell}$ then discard a from D .
-

Stream Sampling. Figure 2. Algorithm for maintaining a sample of all distinct elements of a stream.

computing aggregates over the entire data stream. However, in many cases the interest lies in computing *time-decayed* aggregates of data, where older elements must be discounted. For such aggregates, it is useful to have a sample where a more recent stream element has a higher probability of being included in the sample. For example, a typical *sliding-window* aggregate asks for an aggregate over all data elements that have appeared in a window of the last W elements of a stream – to answer such queries, it is useful to have a random sample of all elements within the current window. The problem with directly using any of the above algorithms to maintain a sample over a sliding window is that elements in the sample may expire, i.e., fall out of the window, and replacing them with enough new sampled elements may not be possible, causing the number of elements within the sample to become too small.

The reservoir sampling algorithm was extended to sliding windows by Babcock et al. [2], who presented probabilistic guarantees on the space taken by their sampling algorithm. Gibbons and Tirthapura [10] have extended the distinct sampling algorithm to sliding windows, by sampling the stream at multiple probabilities, and maintaining a fixed number of the most recent elements at each sampling probability. It is known [4] that for computing decayed aggregates, an arbitrary time-decay function such as polynomial or exponential decay can be reduced to sliding window decay. Thus, for any stream aggregate that can be estimated well using random samples, such as quantiles, frequent elements, distinct counts and sum, the above techniques can be used to estimate time-decayed aggregates over an arbitrary decay function.

Handling Deletions

Thus far, it has been assumed that a data stream is a sequence of additions to a data set (which is so massive that it is too expensive to store it explicitly). More generally, it is necessary to deal with a stream where each element is an *operation* on the data set, which may be an addition or a deletion of one or more elements. On such a stream of add/delete operations, a direct sampling algorithm such as reservoir sampling or distinct sampling may not perform well, since elements that currently belong in the sample may be deleted by a future stream operation, leading to a sample that is too small to give useful estimates. The basic stream sampling algorithms have been extended to handle such “update streams” (sometimes called “dynamic data

streams”) by Ganguly [7], Cormode et al. [5], and Frahling et al. [6].

Key Applications

Estimating Query Selectivity: The selectivity of a query on a database table is defined as the ratio of the number of records that match the query to the total number of records in the table. Suppose that the records appeared as a stream, and it was not possible to store the entire table on the disk (or disk access was too expensive). Then, a stream sampling algorithm can be used to maintain a random sample of all the records. The selectivity of the query on the random sample is an unbiased estimate of the selectivity of the query on the whole stream (i.e., the expected value of the estimate equals the actual selectivity). Of course, the larger the random sample, the more accurate is the estimate. Note that the query can be posed *after* the stream was observed, since the random sampling procedure was not sensitive to the query.

Network Monitoring: In the context of monitoring a TCP/IP network, a “network flow” is a unidirectional set of packets that arrive at the router on the same subinterface, have the same source and destination IP addresses, same transport layer protocol (TCP/UDP), same TCP/UDP source and destination ports and the same type of service (ToS) byte in the IP headers. A network monitoring tool is a software that constantly monitors the flows in a network and helps in network management tasks such as load balancing and fault management. Some network monitoring tools, for example, Random Sampled Netflow (by Cisco), Gigoscope (by AT&T Research), and “Smart Sampling” (by AT&T), provide data for a subset of traffic in a router by processing only a random sample of the packet stream. Traffic sampling substantially reduces consumption of router resources while providing valuable network flow statistics.

Sensor Data Aggregation: A sensor network is a network of resource-constrained embedded devices, capable of computing and sensing, deployed to monitor environmental conditions like temperature, sound, vibration, pressure, light, etc. In a typical scenario, sensors collect readings periodically and send them to a base-station or a local cluster-head where computation/aggregation takes place. Data within a sensor network can be viewed as the union of multiple distributed streams, one per sensor node. Random samples of such distributed data streams can be used

in computing key aggregates of sensor data streams, such as the mean, quantiles, frequent elements, of data. Transmitting the random samples rather than the entire observation stream can lead to savings in communication cost and hence, energy.

Cross-references

- ▶ [AMS Sketch](#)
- ▶ [Data Aggregation in Sensor Networks](#)
- ▶ [Data Sketch/Synopsis](#)
- ▶ [Distributed Streams](#)
- ▶ [Frequency Moments](#)
- ▶ [Randomization Methods to Ensure Data Privacy](#)
- ▶ [Stream Mining](#)

Recommended Reading

1. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. *J. Comput. Syst. Sci.*, 58(1):137–147, 1999.
2. Babcock B., Datar M., and Motwani R. Sampling from a moving window over streaming data. In Proc. ACM-SIAM Symp. on Discrete Algorithms, 2002, pp. 633–634.
3. Chakrabarti A., Cormode G., and McGregor A. A near-optimal algorithm for computing the entropy of a stream. In Proc. ACM-SIAM Symp. on Discrete Algorithms, 2007, pp. 328–335.
4. Cohen E. and Strauss M. Maintaining time-decaying stream aggregates. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003, pp. 223–233.
5. Cormode G., Muthukrishnan S., and Rozenbaum I. Summarizing and mining inverse distributions on data streams via dynamic inverse sampling. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 25–36.
6. Frahling G., Indyk P., and Sohler C. Sampling in dynamic data streams and applications. In Proc. 21st Annual ACM Symp. on Computational Geometry, 2005, pp. 142–149.
7. Ganguly S. Counting distinct items over update streams. *Theor. Comput. Sci.*, 378(3):211–222, 2007.
8. Gibbons P. Distinct sampling for highly-accurate answers to distinct values queries and event reports. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 541–550.
9. Gibbons P. and Tirthapura S. Estimating simple functions on the union of data streams. In Proc. ACM Symp. on Parallel Algorithms and Architectures, 2001, pp. 281–291.
10. Gibbons P. and Tirthapura S. Distributed streams algorithms for sliding windows. *Theor. Comput. Syst.*, 37:457–478, 2004.
11. Manku G.S. and Motwani R. Approximate frequency counts over data streams. In Proc. of the 28th Int. Conf. on Very Large Data Bases, 2002, pp. 346–357.
12. Manku G.S., Rajagopalan S., and Lindsay B.G. Random sampling techniques for space efficient online computation of order statistics of large datasets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 251–262.

13. Olken F. and Rotem D. Random sampling from databases – a survey. *Stat. Comput.*, 5(1):43–57, 1995.
14. Pavan A. and Tirthapura S. Range-efficient counting of distinct elements in a massive data stream. *SIAM J. Comput.*, 37(2):359–379, 2007.
15. Vitter J.S. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, 1985.

Stream Similarity Mining

ERIK VEE

Yahoo! Research, Silicon Valley, CA, USA

Synonyms

[Distance between streams](#); [Datastream distance](#)

Definition

In many applications, it is useful to think of a data-stream as representing a vector or a point in space. Given two datastreams, along with a distance or similarity measure, the distance (or similarity) between the two streams is simply the distance (respectively, similarity) between the two points that the datastreams represent. Due to the enormous amount of data being processed, datastream algorithms are allowed just a single, sequential pass over the data; in some settings, the algorithm may take a few passes. The algorithm itself must use very little memory, typically polylogarithmic in the amount of data, but is allowed to return approximate answers.

There are two frequently used datastream models. In the *time series model*, a vector, \vec{x} , is simply represented as data items arriving in order of their indices: x_1, x_2, x_3, \dots . That is, the value of the i th item of the stream is precisely the value of the i th coordinate of the represented vector. In the *turnstile model*, each arriving item signals an update to some component of the represented vector. So item (i, a) indicates that the value of the i th component of the vector is increased by a . For this reason, datastream items are typically written in the form $(i, x_i^{(j)})$ to indicate that this is the j th update to the i th component of the represented vector. The value of x_i is then the sum of $x_i^{(1)} + x_i^{(2)} + \dots$ over all such updates. The update values may be negative; the special case when they are restricted to be nonnegative is sometimes called the *cash register model*.

One of the most commonly used measures for datastream similarity is the L_p distance between two

streams, for $p \geq 0$. As in the standard definition, the L_p distance between points \vec{x}, \vec{y} (hence, between streams representing those points) is defined to be $\sum_i |x_i^p - y_i^p|^{1/p}$. In the case that $p = 0$, the L_0 distance (sometimes called the Hamming distance) is taken to be the number of i such that $x_i \neq y_i$. For $p = \infty$, the L_∞ distance is $\max_i |x_i - y_i|$. Other measures include the Jaccard similarity, the edit distance, the earth-mover's distance, and the length of the longest common subsequence between the streams (viewed as sequences).

Historical Background

Although the earliest datastream-style algorithms were discovered some 30 years ago [11], the current resurgence of interest in datastreams began with the seminal paper of Alon et al. [2] in 1996. Implicit in their work is an algorithm for estimating the L_2 distance between streams. In 1999, Feigenbaum et al. [10] developed a datastreaming algorithm to approximate the L_1 distance between two streams. Building on this, Indyk [12] gave datastreaming algorithms to approximate the L_p distance between two datastreams, for all $p \in (0,2]$, utilizing the idea of p -stable distributions. Later, Cormode et al. [7] demonstrated an efficient algorithm for approximating the L_0 distance (i.e., Hamming distance). Sun and Saks [15] provide lower bounds for approximating L_p , for $p > 2$ (and including $p = \infty$), showing no datastream algorithm working in polylogarithmic space can approximate the L_p distance between two streams within a polylogarithmic factor. (The bounds are even stronger for p much larger than 2.)

Datar et al. [8] studied the sliding window model for datastreams, producing an algorithm that approximates the L_p distance between two windowed datastreams. Work by Datar and Muthukrishnan [9] gave an algorithm for approximating the Jaccard similarity between two datastreams in the sliding window model.

Foundations

Estimating the L_2 Distance

In their seminal paper, Alon et al. [2] provide a method for estimating F_2 , the second frequency moment, of a datastream. As observed in [10,1], this method can easily be extended to produce a datastream algorithm to approximate the L_2 distance. The ideas are briefly outlined below.

Throughout, the datastreams considered have length n . For $i = 1, 2, \dots, n$, the variable X_i is defined to

be an i.i.d. (independent and identically distributed) random variable taking on the value -1 or 1 with equal probability. Of course, a datastream algorithm cannot maintain all the values of each of the random variables in memory. This will be accounted for later; for now, an algorithm is presented assuming that there is random access to these values.

The datastreams vectors are represented in the turnstile model; (x_1, \dots, x_n) denotes the accumulated values of the first stream, and (y_1, \dots, y_n) denotes the accumulated values in the second stream. The algorithm simply maintains the value of $\sum_{i=1}^n X_i \cdot (x_i - y_i)$. This value is straightforward to maintain: If an item $(i, x_i^{(j)})$ arrives for some i, j , the value $X_i \cdot x_i^{(j)}$ is added to it. If an item $(i, y_i^{(j)})$ arrives, the value $X_i \cdot y_i^{(j)}$ is subtracted.

The algorithm focuses on the expected value of *the square* of this quantity:

$$\begin{aligned} E\left[\left(\sum_{i=1}^n X_i \cdot (x_i - y_i)\right)^2\right] \\ = E\left[\sum_{i=1}^n X_i^2 \cdot (x_i - y_i)^2\right. \\ \quad \left.+ \sum_{i \neq j} X_i X_j \cdot (x_i - y_i)(x_j - y_j)\right] \\ = \sum_{i=1}^n (x_i - y_i)^2, \end{aligned}$$

where the last equality follows since $E[X_i] = 0$ and $X_i^2 = 1$ for all i , and all the random variables are independent. But this quantity is just the square of the L_2 distance between the two streams. Hence, the problem amounts to obtaining a good estimate of this expected value.

To do so, the above algorithm is run in parallel k times, for $k = \theta(1/\epsilon^2)$. That is, it maintains the value $\sum_{i=1}^n X_i \cdot (x_i - y_i)$ for k different random assignments of the X_i . The algorithm then takes the average of their squares. For a given run t , this value is denoted $v^{(t)}$. To further ensure that the algorithm does not obtain a spurious estimate, the procedure is repeated ℓ times, for $\ell = \theta(\log(1/\delta))$. The algorithm then takes the median value over $\{v^{(1)}, v^{(2)}, \dots, v^{(\ell)}\}$. A standard application of Chebyshev's Inequality shows that this estimates the square of the L_2 distance within a $(1 + \epsilon)$ factor with probability greater than $1 - \delta$. (In total, this method maintains $k\ell$ values in parallel.)

Unfortunately, the procedure as described above produces and maintains values for n random variables. (In fact, due to the parallel repetitions, it actually needs $k\ell n$ random variables.) However, the technique only needed these variables to be four-wise independent. (Two-wise independence is needed for the expected value to be an unbiased estimator of the square of the L_2 distance; four-wise independence implies that the variance is small.) Hence, these fully independent random variables can be replaced with four-wise independent random variables, which is necessary for Chebyshev's Inequality to hold. These random variables can be pseudorandomly generated on the fly; the datastream algorithm thus only needs to remember a logarithmic-length seed for the pseudorandomly generated values. The full details are omitted here.

Estimating the L_p Distance: p -Stable Distributions

In 2000, Indyk [12], using many of the ideas in [2, 10], extended the results to produce datastream algorithms for approximating the L_p distance between streams, for all $p \in (0, 2]$. (Feigenbaum et al. were the first to produce a datastream algorithm for L_1 distance; their technique relied on their construction of pseudorandomly generated “range-summable” variables that were four-wise independent. Although similar in flavor to the result of [2], it is somewhat more complicated.) For convenience, the algorithm outlined below details the method for approximating the L_p norm of a single vector. Note, however, that in the turnstile model, it is a simple matter to produce the L_p distance between two streams (by simply negating all of the values in the second stream and finding the norm of their union). Indyk's method uses random linear projections, and relies on the notion of *p-stable distributions*.

A distribution \mathcal{D} is *p-stable* if for all k real numbers a_1, \dots, a_k if X_1, \dots, X_k are i.i.d random variables drawn from distribution \mathcal{D} , then the random variable $\sum_i a_i X_i$ has the same distribution as $(\sum_i |a_i|^p)^{1/p} X$ for random variable X with distribution \mathcal{D} . There are two well-known *p*-stable distributions. The *Cauchy distribution*, with density function $\mu_C(x) = \frac{1}{\pi} \frac{1}{1+x^2}$, is 1-stable. The *Gaussian distribution*, with density function $\mu_G(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$, is 2-stable. Although closed-form functions are not known for *p*-stable distributions for $p \neq 1, 2$, Chambers et al. [4] provide a method for generating *p*-stable random variables for all $p \in (0, 2]$. Throughout the rest of this discussion, \mathcal{D} denotes a *p*-stable distribution, for some fixed p .

The method for approximating the L_p norm of a stream will now be outlined. As previously noted, this is easily modified to give the L_p distance between two streams. Throughout, the vectors are represented as in the turnstile model, and (z_1, \dots, z_n) denotes the vector represented by the datastream. As in the previous section, the n i.i.d. random variables X_1, \dots, X_n are generated first, this time drawn from *p*-stable distribution \mathcal{D} . A brief discussion of how to reduce the number of these variables appears later.

The algorithm simply maintains the value $\sum_i X_i z_i$. Again, these values are easy to maintain: If item $(i, z_i^{(j)})$ appears for some i, j , the algorithm adds the value $X_i z_i^{(j)}$ to the sum. As in the previous section, the algorithm gains better accuracy by repeating the procedure multiple times in parallel; in this case, the algorithm runs the procedure k times in parallel, for $k = \theta(\frac{1}{\epsilon^2} \log(1/\delta))$. The value of $\sum_i X_i z_i$ obtained in the ℓ -th run using this procedure is denoted $Z^{(\ell)}$.

The value $Z^{(\ell)}$ is a random variable itself. Since \mathcal{D} is *p*-stable, it is the case that $Z^{(\ell)} = X^{(\ell)} \cdot (\sum_i |z_i|)^{1/p}$ for some random variable $X^{(\ell)}$ drawn from \mathcal{D} . Then the output of the algorithm is

$$\frac{1}{\gamma} \text{median}\{|Z^{(1)}|, \dots, |Z^{(k)}|\},$$

where γ denotes the median value of $|X|$, for X a random variable distributed according to \mathcal{D} . (The absolute value is taken for technical reasons. For instance, the median value of X is 0 when \mathcal{D} is the Gaussian distribution, while the median value of $|X|$ is strictly greater than 0.) The value of the median of $\{|Z^{(1)}|, \dots, |Z^{(k)}|\}$ is $(\sum_i |z_i|)^{1/p}$ times the median of $\{|X^{(1)}|, \dots, |X^{(k)}|\}$. Hence, the above output is an approximation of $(\sum_i |z_i|)^{1/p}$, i.e., the L_p norm of the datastream, as needed. A more careful argument shows that this estimate is within a multiplicative factor $(1 \pm \epsilon)$ of the true L_p norm, with probability greater than $1 - \delta$.

As in the previous section, Indyk observes that rather than storing the values of n i.i.d random variables, the values can be generated on the fly, using pseudorandom generators. The details are omitted here.

Cormode et al. [7] investigate the problem of estimating the L_0 norm. One of their key technical observations is that the L_p norm is a good approximation of the L_0 norm of the stream, for p sufficiently small. (In particular, they show the $p = \epsilon \log M$ is sufficient, where M is the maximum absolute value of any item in the

stream.) Thus, the Hamming distance between two streams can be approximated using the same general algorithm that was described above.

Approximating Jaccard Similarity: Min-Wise Hashing

Another useful similarity measure between two streams is their *Jaccard similarity*. Given two data-streams in the time-series model, a_1, a_2, \dots, a_n and b_1, b_2, \dots, b_n denote their respective vectors. Further, A (and B) denotes the set of distinct elements appearing in the first stream (respectively, the second stream). The Jaccard similarity between the streams is given by $|A \cap B|/|A \cup B|$.

The first explicit study of the Jaccard similarity between two streams was given by Datar and Muthukrishnan [9]. Their paper examined the sliding window model, which is discussed further in the next section. However, a datastream algorithm in the standard model was given implicitly in the work of Cohen et al. [6], although the notion of datastreams is never mentioned in the paper.

The major technical tool uses *min-wise hashing*, or min-hashing [3,5]. For every subset A of $[n]$, the *min-hash* for A (with respect to π), denoted $h_\pi(A)$, is defined to be $h_\pi(A) = \min_{i \in A} \{\pi(i)\}$, where π denotes a permutation on $[n] = \{1, \dots, n\}$. The wonderful property of the min-hash is that, when π is chosen uniformly at random from the set of all permutations on $[n]$, for any two subsets A, B of $[n]$, it is the case that

$$\Pr[h_\pi(A) = h_\pi(B)] = \frac{|A \cap B|}{|A \cup B|}.$$

This suggests the following algorithm.

The algorithm chooses π uniformly at random from the set of permutations on $[n]$. (The fact that storing π take $\theta(n \log n)$ space will be discussed momentarily.) For the first stream, the algorithm finds the value $h_\pi(A) = \min_{i \in A} \{\pi(i)\}$, where A is the set of distinct elements occurring in the first stream. This is simple to do in a datastreaming fashion: as each new a_j appears, the algorithm updates the min value if $\pi(a_j)$ is smaller than the min seen so far. Likewise, for the second stream, the algorithm finds the value $h_\pi(B)$, where B is the set of distinct elements occurring in the second stream. From the above, the probability that the two values are equal is precisely the Jaccard similarity between the two streams.

Of course, to obtain an accurate estimate of this probability, the algorithm needs to run the procedure

multiple times. In this case, it will run the procedure in parallel k times, each with an independently chosen random permutation. (Here, $k = O(\varepsilon^{-3} \log(1/\delta))$.) The value ρ is defined to be the fraction of times (out of k) that the min values for the two streams coincide. That is, if π_1, \dots, π_k are the k independently chosen random permutations, then

$$\rho = \frac{1}{k} \cdot \# \{j : h_{\pi_j}(A) = h_{\pi_j}(B)\}.$$

It is shown in [11] that with probability at least $1 - \delta$, the value ρ approximates the Jaccard similarity within multiplicative factor $(1 \pm \varepsilon)$.

In order for the above algorithm to be useable in a datastreaming context, it must be able to generate and store the necessary random permutations in small space. This is done using *approximately* min-wise independent hash functions. Although this introduces additional error, it can be done in small space and time. The reader is referred to [13] for more details.

Sliding Windows

In many applications, the data from streams becomes outdated or unnecessary quickly. To help understand this scenario better, researchers have proposed the *sliding window* model of datastreams. Here, the algorithm must maintain statistics (e.g., stream similarity), using only the last N items from the stream, for some N . This causes additional complications, since as each new item comes in, an old item is removed. Since memory is limited, algorithms cannot track which of these old items is disappearing. Still, there are datastream algorithms for both L_p distance and Jaccard similarity in the sliding window model.

In [8], Datar et al. define the sliding window model, and give a datastream algorithm for approximating the L_p distance between two streams (as well as several other datastream algorithms). Their technique uses what they call an *exponential histogram*. The histogram partitions the last N items (i.e., those items in the sliding window) into buckets; the last bucket may in fact contain items older than the last N . Each bucket maintains the necessary statistics for the items it contains. For instance, a bucket containing the items a_s, a_{s+1}, \dots, a_t would hold the L_p -sketch for those items. (Due to memory constraints, the bucket cannot actually maintain the values of all the items it holds.)

As new items come in, the algorithm merges old buckets to maintain the histogram structure, creating new buckets only for newly encountered items. The last bucket will eventually contain only items that do not appear in the N most recent, and will be removed from the histogram at this time. Datar et al. observe that the additional error in this windowed model, beyond that of the standard model, comes from the fact that the last bucket may contain items that are no longer in the N -item window. But the structure of the exponential histogram ensures that this error is not too large. Hence, they provide a general method for translating a wide range of datastream algorithms into windowed-datastream algorithms.

Datar and Muthukrishnan [9] study the problem of approximating the Jaccard similarity of two streams in the sliding window model. As in the non-windowed version, they use min-hashing as a primary tool. The main complication in the sliding window model is that maintaining the minimum value over a sliding window is hard. At a given time step t , the algorithm needs to know the value $\min_{i=t,\dots,t-N+1} \{\pi_j(a_i)\}$, where π_j is a permutation chosen by the datastream algorithm in the standard model. Their solution is to maintain the value $\pi_j(a_i)$ for every $\text{relevant } i = t, \dots, t - N + 1$. For instance, if $\pi_j(a_i) > \pi_j(a_{i+s})$ for some $s > 0$, then the value $\pi_j(a_i)$ will never be the minimum over the sliding window at any time; hence, it may be discarded. (Here, item a_i occurs earlier than a_{i+s} , thus item a_i will move out of the window before a_{i+s} .) Amazingly, with high probability, the number of relevant values that need to be maintained is at most $O(\log n)$. Hence, the standard datastream algorithm can be adapted to the sliding window model, using small space.

Lower Bounds for Stream Distance

The major technique for proving lower bounds utilize reductions from communication complexity. Here, only sketches of the very high level ideas are presented, with some of the main results cited.

An often-used communication complexity problem is DISJOINTNESS: Alice is given a set, A , and Bob is given a set, B . Neither knows what the other set is. They must communicate with each other by sending messages back and forth, until they decide whether $A \cap B$ is nonempty. (They are allowed to decide ahead of time the protocol they will use to communicate messages.) It has been shown that if the size of A and B is $\theta(n)$, the communication complexity (i.e., the number of bits

that must be communicated in the worst case) is also at least $\theta(n)$ [14].

A datastream algorithm that calculates the distance between two streams can provide the basis for a communication complexity algorithm. A typical reduction gives a method for Alice to transform her set A into a datastream (without looking at set B). Likewise, the reduction gives a method for Bob to transform B into a datastream, without looking at A . Finally, the reduction guarantees that Alice's datastream and Bob's datastream are close if and only if $A \cap B$ is non-empty. Then Alice can begin running the datastream algorithm on her datastream. When it has processed her stream, the algorithm will have some memory bits indicating its current state. Alice sends a message to Bob, telling him that state. Bob can then finish running the datastream algorithm on his own datastream. If the algorithm indicates that the two streams are close, he knows $A \cap B$ is nonempty; otherwise, he knows that $A \cap B = \emptyset$ (and may communicate this to Alice in one bit). Hence, Alice and Bob have solved their communication complexity problem. Since the original communication complexity problem took at least $\theta(n)$ bits, the datastream algorithm must also use at least this much memory. (In this case, showing that it cannot be space efficient.)

There is, of course, a great deal of technical work in providing the proper reductions; the difficulties are even greater when showing lower bounds for approximations. However, building on these ideas, Saks and Sun [15] show that approximating the L_∞ distance between two datastreams is impossible to do in sub-linear space. In fact, their work shows that approximating within factor $n^{O(\epsilon)}$ the L_p distance for any $p \geq 2 + \epsilon$ requires space at least $n^{O(\epsilon)}$. For p close to 2, this has very little practical implications, but the bounds become more meaningful for large p . Much simpler reductions show the impossibility of space-efficient datastream algorithms for approximating the length of the longest common subsequence between two datastreams (viewed as sequences).

Key Applications

Tracking Change in Network Traffic

The datastream algorithms outlined above allow one to take an entire day of network traffic and synopsize it using a small sketch. It is then possible to measure how different traffic is from day-to-day. Large changes in

the network traffic can signal denial of service attacks or worm infestations.

Query Optimization

Most query-optimization techniques utilize data statistics to produce better plans. The L_2 norm is a useful measure for approximating join sizes, while the L_0 norm gives the number of distinct items in the stream.

Processing Genetic Data

Since genetic data consists of millions or billions of base pairs for an individual, it is useful to think of them as streams of data. The similarity of two base-pair sequences is a fundamental concept.

Data Mining

Often individual entities are represented by massive streams of data (e.g., phone calls from a large company, or IP addresses of users visiting a given web site, or items bought at a grocery store). Estimating the similarity between these streams can be a useful tool for identifying similar entities. As one example, it is possible to determine which web sites are most similar to each other, based on the IP addresses of their visitors.

Cross-references

- ▶ [Approximation and Data Reduction Techniques](#)
- ▶ [Stream Data Management](#)
- ▶ [Stream Mining](#)

Recommended Reading

1. Alon N., Gibbons P., Matias Y., and Szegedy M. Tracking join and self-join sizes in limited storage. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 10–20.
2. Alon N., Matias Y., and Szegedy M. The space complexity of approximating the frequency moments. In Proc. 28th ACM Symp. on Theory of Computing, 1996, pp. 20–29.
3. Broder A., Charikar M., Frieze A., and Mitzenmacher M. Min-wise independent permutations. In Proc. of the 30th ACM Symp. on Theory of Computing, 1998, pp. 327–336.
4. Chambers J.M., Mallows C.L., and Stuck B.W. A method for simulating stable random variables. *J. Am. Stat. Assoc.*, 71:340–344, 1976.
5. Cohen E. Size-estimation framework with applications to transitive closure and reachability. *J. Comput. Syst. Sci.*, 55:441–453, 1997.
6. Cohen E., Datar M., Fujiwara S., Gionis A., Indyk P., Motwani R., and Ullman J. Finding interesting associations without support pruning. In Proc. 16th International Conf. on Data Engineering, 2000.

7. Cormode G., Datar M., Indyk P., and Muthukrishnan S. Comparing data streams using hamming norms. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 335–345.
8. Datar M., Gionis A., Indyk P., and Motwani R. Maintaining stream statistics over sliding windows. In Proc. 13th Annual ACM-SIAM Symp. on Discrete Algorithms, 2002, pp. 635–644.
9. Datar M. and Muthukrishnan S. Estimating rarity and similarity on data stream windows. In Proc. 10th European Symp. on Algorithms, 2002.
10. Feigenbaum J., Kannan S., Strauss M., and Viswanathan M. An approximate l_1 -difference algorithm for massive data streams. In Proc. 40th Annual Symp. on Foundations of Computer Science, 1999.
11. Flajolet P. and Martin G. Probabilistic counting. In Proc. 24th Annual Symp. on Foundations of Computer Science, 1983, pp. 76–82.
12. Indyk P. Stable distributions, pseudorandom generators, embeddings and data stream computation. In Proc. 41st Annual Symp. on Foundations of Computer Science, 2000, pp. 189–197.
13. Indyk P. A small approximately min-wise independent family of hash functions. *J. Algorithm.*, 38:84–90, 2001.
14. On the distributional complexity of disjointness. *J. Comput. Sci. Syst.*, 2, 1984.
15. Saks M. and Sun X. The space complexity of approximating the frequency moments. In Proc. 34th ACM Symp. on Theory of Computing, 2002.

Streaming Algorithm

- ▶ [One-Pass Algorithm](#)

Streaming Applications

YANIF AHMAD, UĞUR ÇETINTEMEL
Brown University, Providence, RI, USA

Synonyms

[Stream-oriented applications](#); [Continuous query processing applications](#)

Definition

Streaming applications typically involve the processing of continuous data streams for the purposes of filtering, aggregation, correlation, transformation, pattern matching and discovery, and domain-specific temporal analytics. These applications often require such continuous processing to be performed with both high throughput and low latency, and are able to

tolerate approximate results and forego some of the persistence requirements of standard database transaction processing applications.

Key Points

A large fraction of streaming applications are monitoring oriented: they involve the tracking of events or activities to identify and act upon situations (or patterns) of interest, either manually or automatically. This so-called “sense and respond” model requires query results to be generated in real-time (meaning low latency) as results lose their utility over time. As such, persistence of all the input data is often not an application requirement, unlike in traditional database applications. Thus, most input data can be simply discarded or, alternatively, asynchronously recorded for archival needs. As on-line data sources proliferate and consolidate, streaming applications need to deal with increasingly higher volume streams, which makes real-time operation especially challenging.

A flagship stream processing application is automated trading, which continually watches market streams (bids and asks) from financial feed providers (e.g., Reuters), evaluating sophisticated real-time patterns over them to identify arbitrage opportunities and automatically act on them. For automated trading, the desired processing latencies are in milliseconds (and continually decreasing) and the estimated peak input data rates are in 170,000 messages/s (as of July 2006), with rates roughly doubling every year [1]. Network monitoring is another application that has stringent real-time response needs under high data volumes: network elements (e.g., routers, gateways) are instrumented to log, summarize and forward traffic data, which is then analyzed to identify and automatically respond to online security attacks (e.g., denial of service attacks) and QoS problems (e.g., SLA violations). Another early streaming application is event detection in MMORPGs, where the virtual game world is continually monitored to identify oddities, semantic bugs and cheats.

Overall, streaming applications abound in various verticals including:

- Financial services: automated trading, market feed processing (cleaning, smoothing, and translation), smart order routing, real-time risk management and compliance (MiFID, RegNMS)

- Government and military: surveillance, intrusion detection and infrastructure monitoring, battlefield command and control
- Telecommunications: network management, quality of service (QoS)/service level agreement (SLA) management, fraud detection
- Web/E-business: click-stream analysis, real-time customer experience management (CEM)
- Entertainment: online gaming (online cheat, bug detection)
- Retail and logistics: automated supply-chain management
- Healthcare: patient monitoring
- Energy: power-grid/pipeline monitoring and control

Cross-references

- ▶ [Data Stream Management Architectures and Prototypes](#)
- ▶ [Stream-oriented Query Languages and Operators](#)

Recommended Reading

1. Options Price Reporting Authority (OPRA) Traffic Projections, http://www.opradata.com/specs/projections_2005_2006.pdf

Streaming Database Systems

- ▶ [Event Driven Architecture](#)

Stream-Oriented Applications

- ▶ [Streaming Applications](#)

Stream-Oriented Query Languages and Operators

MITCH CHERNIACK¹, STAN ZDONIK²

¹Brandeis University, Waltham, MA, USA

²Brown University, Providence, RI, USA

Synonyms

[Continuous query languages](#)

Definition

Many research prototypes and commercial products have emerged in the new area of stream processing. All of these systems support a language for specifying queries. A fundamental difference between a stream query language and a conventional query language like SQL is that stream queries are not one-time computations, but rather, they continue to produce answers as new tuples arrive on one or more input streams. Thus, queries are registered with the system and answers continue to evolve over time. This new assumption is crucial to understanding some of the technical differences that arise in stream query languages.

Most stream query languages try to extend SQL in one way or another. The form of these extensions can be either a purely textual extension of SQL or GUI, through which users can construct dataflow diagrams that connect extended versions of relational operators. These days, many systems provide both.

The most fundamental addition to a stream query language over their relational counterparts is the notion of a window. Windows produce finite structures (i.e., tables) from infinite structures (i.e., streams). Much of the technical detail of a streaming data model revolves around the specifics of how windows are formed. This will be discussed in detail below.

One of the biggest technical challenges in the implementation of such a stream query language is the ability to produce answers with minimum latency. The latency requirement is a reaction to the kinds of applications that stream processing was invented to address. In broad terms, these applications have to do with monitoring conditions on the input streams. Typically, in this setting, the value of the answer decays quickly.

The main thrust of this article is on the technical choices that must be faced by anyone who designs a stream query language. The main concepts with examples from major systems will be illustrated. There are also many related technologies and associated languages (e.g., XML streams, temporal databases, active databases) that are not discussed in this article. The focus is on extensions to the relational model and relational languages (e.g., relational algebra, SQL) that incorporate streams.

Historical Background

There has been significant work on stream query languages in the past. The academic languages include:

CQL [3,10] (from the STREAM project out of Stanford),

SQuAl [1,6] (from the Aurora/Borealis project out of Brandeis, Brown and MIT), and

ESL [4,13] (from the Atlas project out of UCLA).

The commercial languages include:

StreamSQL [11] (from Streambase),

CCL [9] (from Coral8),

EQL [7] (from Esper), and

StreaQuel [5,8] (from Truviso).

The commercial stream query languages are, in many cases, derived from the academic languages; StreamSQL is derived from SQuAl and CQL, CCL is derived from CQL, and StreaQuel is derived from a language of the same name from the Telegraph CQ project out of UC Berkeley. The material from which information about these languages was gleaned is listed at the end. Especially in the case of commercial languages, published documentation on these languages is sometimes incomplete. Thus, there may be omissions in the descriptions of the features of these languages that follow. For example, documentation on StreaQuel is especially scant and so the description for this language is likely to be incomplete.

Foundations

Stream query languages (both academic and commercial) primarily differ in how they approach the most fundamental requirements of stream processing. These requirements include:

1. *Language Closure*: Are the language's operators closed under streams? Or does the language support operators that convert streams to relations and/or relations to streams? The approach a language takes to closure reveals a lot about how tightly integrated is the language with a relational query language such as SQL.
2. *Windowing*: Does the language support first-class windows? (I.e., are windows namable, sharable and queryable?). Or are windows internal to the definition of stateful operations. And what kinds of windows can be expressed in the language?
3. *Correlation*: Does the language provide a way to correlate tuples (events) arriving on a stream with historical data, and with tuples arriving on a separate stream?
4. *Pattern Matching*: Does the language have a way to identify interesting subsequences of tuples on one or more streams?

Language Closure: The simplicity of the relational algebra/calculus is largely due to the closure property that says that all inputs and outputs of relational queries are relations. Beyond simplifying the type system, closure also ensures that the output of any one query can be input to another.

The various stream query languages compared here approach closure in one of two ways. Languages such as SQuAl and EQL define a language that is closed under streams. That is, every operator in these two languages accepts one or more streams as input and produces streams as output. (SQuAl includes two operators that have the side-effect of accessing a relation (*WRITESQL* and *READSQL*), but both return stream outputs.) The other languages in this list include both streams and relations in their type system. For example, CQL includes no stream-to-stream operations. Instead, operation on a stream demands that it first be converted into a relation (via windowing). Thereafter, all query operations are relation-to-relation operations as in SQL. If the desired result is a stream, the specialized operators *ISTREAM*, *DSTREAM* and *RSTREAM* produce stream outputs from relational inputs by (essentially) returning the log of changes to the relations in the order in which they occur. The other languages in this list (StreamSQL, CCL and StreaQuel) largely follow the CQL model, except that queries in these languages that contain exactly one unwindowed stream in the FROM clause are considered to be stream-to-stream. For example, in these languages, the query,

```
SELECT *
FROM S
WHERE p
```

returns a stream if S is a stream. In CQL, this query is assumed to be a syntactic shorthand for,

```
SELECT ISTREAM *
FROM S [∞]
WHERE p
```

and thus also returns a stream, but only as a result of first windowing S into a relation (see the subsection below on windowing) and then converting the result back into a stream.

Many of the query languages that include both relations and streams in their data model also define query operations that produce relations from streams and streams from relations. As mentioned previously,

CQL produces streams from relations using windows, and the specialized operations *ISTREAM*, *DSTREAM* and *RSTREAM* produce relations from streams. StreamSQL, CCL and StreaQuel all include additional operations for producing relations from windows (*INSERT*, *UPDATE* and *DELETE*), which update pre-specified relations with the arrival of each tuple on a stream in the same way that the equivalent SQL operations update a relation with an individual tuple. StreaQuel and StreamSQL also support *ISTREAM* and *DSTREAM* (and in the case of StreaQuel, *RSTREAM*) operations that produce streams from relations, as in CQL. No such operations are described in the publicly available literature on CCL as of the time of this writing.

[Table 1](#) summarizes the closure properties of the academic and commercial languages studied in this report, as well as the operations the languages support (if any) for producing relations from streams and streams from relations. Note that all stream languages support windowing as a means of producing a relation from a stream. However, some of these languages (e.g., SQuAl, ESL, EQL) are still considered to be closed under streams because their window definitions are internal to the query's operation and not output.

Windowing: All stream query languages have some form of windowing to convert infinite streams into automatically maintained, time-varying relations. But different query languages vary in whether they support first-class windows, and in terms of the features of window definition that they support. [Table 2](#) summarizes how various stream query languages support windows.

A first-class window is a window that can be named, shared and independently queried. Put simply, a query language supports first-class windows allows a window to be named and defined as the result of a statement in the query language, and subsequent queries can then access this window. Windows are not first-class if they are defined as part of a query, but are not visible outside of the execution of that query. First-class windows are typically supported in query languages that are closed under both streams and relations (i.e., CQL, StreamSQL, CCL and StreaQuel). Languages such as SQuAl, EQL and ESL that are closed under streams define windows internally within queries.

All windows are characterized as having a certain size, and advancing in some way (i.e., adding new

Stream-Oriented Query Languages and Operators. Table 1. Closure properties of stream query languages

Features		Closed under	Operations			
			Stream-to-stream		Stream-to-relation	Relation-to-stream
Academic Languages	CQL (STREAM)	Streams, Relations	No, but many queries assume use of ISTREAM		Windows	ISTREAM, DSTREAM, RSTREAM
	SQuAl (Aurora)	Streams	Default		Windows, WRITESQL	READSQL
	ESL (Atlas)	Streams	Default		Windows	-
Commercial Languages	StreamSQL (Streambase)	Streams, Relations	All queries with unwindowed stream in FROM clause		Windows, INSERT, UPDATE, DELETE	ISTREAM, DSTREAM
	CCL (Coral8)	Streams, Relations	All queries with unwindowed stream in FROM clause		Windows, INSERT, UPDATE, DELETE	-
	EQL (Esper)	Streams	Default		Windows	-
	StreaQuel (Truviso)	Streams, Relations	Default (from implicit use of ISTREAM and RSTREAM)		Windows, Active Tables	?

Stream-Oriented Query Languages and Operators. Table 2. Windowing support in stream query languages

Features		First-class	Windows (sizing)				Windows (movement)		Windows (other features)	
			Row-based	Time-based	Value-based	Sliding	Tumbling	Sampling	Top-K	Landmark
Academic Languages	CQL (STREAM)	Yes	Yes	Yes	No	Yes	Yes	X% SAMPLE	-	-
	SQuAl (Aurora)	No	Yes	Yes	Yes	Yes	Yes	RESAMPLE	BSORT	-
	ESL (Atlas)	No	Yes	Yes	No	Yes	Yes	Definable with UDA's	Definable with UDA's	Definable with UDA's
Commercial Languages	StreamSQL (Streambase)	Yes	Yes	Yes	Yes	Yes	Yes	-	EVICT MIN/MAX	-
	CCL (Coral8)	Yes	Yes	Yes	No	Yes	Yes	-	KEEP LARGEST/SMALLEST	Yes (KEEP FOR DURATION)
	EQL (Esper)	No	Yes	Yes	No	Yes	Yes	-	-	-
	StreaQuel (Truviso)	Yes	Yes	Yes	No	Yes	Yes	?	?	Yes

tuples and deleting old ones as new tuples arrive on a stream). Though each language has its own syntax for specifying how to size and advance a window, all of the languages discussed looked at support windows whose size is “row-based” (i.e., defined by the number of tuples contained in the window) or “time-based” (i.e., defined by the maximum time interval between any two tuples in the window). SQuAl and StreamSQL

also support “value-based” windows over streams whose tuples are known to arrive in ascending order on some data field of the tuple. Value-based windows specify the maximum difference in value of that attribute between tuples in the same window. All languages studied in this article looked at support sliding by some query-specified amount (expressed in number of tuples, increment in time, or (in the case of SQuAl

and StreamSQL), increment in value of the attribute over which the windowed stream is ordered. As well, all of the languages discussed here support “tumbling windows,” which are windows that contain tuples that belong to exactly one window.

The last 3 columns of [Table 2](#) show some of the window properties that are supported by some but not all of the query languages discussed here. For example, both CQL and SQuAl support a form of sampling to determine the contents of a window. In the case of CQL, a window defined with *SAMPLE* will consist of a subset of the most items that have arrived on the windowed stream. In the case of SQuAl, a window defined with *RESAMPLE* will use interpolation with a user-supplied function to fill-in missing values from the windowed stream. SQuAl, StreamSQL and CCL all support “top-k” (or “bottom-k”) windows, which at any point in time, contain those tuples that have arrived on the windowed stream that have the maximum (minimum) values of some specified attribute. And CCL and StreaQuel both support “Landmark Windows”; so-named because these are windows which are fixed at the start-point with an end-point that advances as tuples arrive on the windowed stream. ESL has expressive user-defined aggregate support whereby aggregates are defined with SQL statements on an internal table that specify how to initialize, increment and return a final result, and this mechanism could be used to specify each of the window types described here.

Correlation: A key component of any stream query language is its support for correlating tuples appearing on a stream with either a repository of historical data, or with the tuples appearing on another stream. As [Table 3](#) shows, all of the query languages studied here support both forms of correlation, though in different ways.

The correlation of tuples on a stream with historical data are expressed in most languages by allowing exactly one stream to appear in a stream query’s FROM clause. Then, either periodically (e.g., CQL) or upon the arrival of each tuple on this stream (e.g., StreamSQL), the query is reevaluated using the tuple(s) that have arrived on the stream since the last time the query was evaluated. ESL, StreamSQL, CCL, EQL and StreaQuel all allow FROM clauses to include one unwindowed stream for this purpose. A CQL query containing one unwindowed stream implicitly windows that stream using the “NOW” size directive, which says to create a window with all tuples that have arrived on the stream since the last time when all queries were reevaluated. Thus, the FROM clause of a CQL query always consists solely of relations including those resulting from windowing streams. SQuAl, which has a graphical rather than SQL-like notation, uses the operation, *READSQL* to correlate stream data with historical data.

Most query languages correlate the tuples on two streams by first windowing at least one (or in case of CQL, both) of the streams. The resulting query then is

Stream-Oriented Query Languages and Operators. **Table 3.** Correlation support in stream query languages

Features		Correlation	
		Stream-to-relation	Stream-to-stream
Academic Languages	CQL (STREAM)	No, but relation-to-relation correlation with window on most recent tuples in stream (NOW) often has same effect	Must window both streams
	SQuAl (Aurora)	READSQL	JOIN
	ESL (Atlas)	JOIN with Stream, Relation in FROM Clause	Must window one of the streams
Commercial Languages	StreamSQL (Streamparse)	JOIN with Stream, Relation in FROM Clause	Must window one of the streams
	CCL (Coral8)	JOIN with Stream, Relation in FROM Clause	Must window one of the streams
	EQL (Esper)	JOIN with Stream, Relation in FROM Clause	Must window one of the streams
	StreaQuel (Truviso)	JOIN with Stream, Relation in FROM Clause	Must window one of the streams

Stream-Oriented Query Languages and Operators. **Table 4.** Pattern matching support in stream query languages

Features	Pattern matching		
	Multi-stream	Regular expression	
Academic Languages	CQL (STREAM)	-	-
	SQuAI (Aurora)	-	-
	ESL (Atlas)	-	-
Commercial Languages	StreamSQL (Streambase)	Yes (MATCH)	Yes (PATTERN)
	CCL (Coral8)	Yes (MATCHING)	No
	EQL (Esper)	Yes (PATTERN)	Yes (PATTERN)
	StreaQuel (Truviso)	Yes (EVENT clause)	Simple (A B C)

either a join of two relations as in SQL, or a join of an unwindowed stream with a relation as described in the previous paragraph. StreamSQL also has a *GATHER* operation which performs key matching to match each tuple on an input stream with the single tuple it matches on each of the other input streams.

Pattern Matching: Pattern matching is a relatively new addition to stream query languages, and is only supported in the commercial languages examined in this study (though there exist some work in the academic literature on pattern matching on streams such as [12]). Pattern matching in stream query languages can take one of two forms:

Multi-stream pattern matching resembles joins between streams in that it correlates tuples appearing on separate streams according to the order in which they arrive. For example, this form of pattern matching might identify all cases where a particular stock received a bid quote (on a BIDS stream) without a corresponding ask quote (on an ASKS stream) within some specified time period. All of the commercial stream query languages examined here (StreamSQL, CCL, EQL and StreaQuel) support this form of pattern matching.

Single-stream pattern matching looks for a sequence pattern of tuples arriving on a single stream, and typically uses a rich pattern matching language based on regular expressions to express desired patterns. For example, this form of pattern matching might identify all sequence of quotes for a particular company that resulted in an “M-pattern” whereby the stock’s price rises for a time, then falls, then rises again and then falls again [2]. This form of pattern matching is part of a general SQL standard proposal put forth by Streambase, Oracle and IBM, and documented in [13].

A summary of each query language’s support for pattern matching is shown in [Table 4](#).

In short, while there are several query languages for streams with both academic and commercial roots, these languages share much in common that identify the crucial requirements of stream processing. Specifically, all of these languages are closed either under streams, or under streams and relations; all have some notion of windowing to convert streams to relations; all provide ways to correlate tuples on a stream with both historical data and tuples on other streams, and all commercial languages support some form of pattern matching to identify interesting subsequences of tuples from one or more streams.

Key Applications

The applications of stream processing revolve around low-latency monitoring of physical or virtual items or events of interest. Examples include automated trading, network security monitoring, and event detection in massively multiplayer on-line games.

Cross-references

- [Continuous Query](#)
- [Data Stream](#)
- [Event and Pattern Detection over Streams](#)
- [Punctuations](#)
- [Stream Processing](#)
- [Windows](#)

Recommended Reading

1. Abadi D., Carney D., Cetintemel U., Cherniack M., Convey C., Lee S., Stonebraker M., Tatbul N., and Zdonik S. Aurora: a new model and architecture for data stream management. VLDB J., 12(2):120–139, 2003.

2. Anon.s Pattern matching in sequences of rows. SQL Standard Proposal, <http://asktom.oracle.com/tkyte/row-pattern-recognition-11-public.pdf>, March, 2007.
3. Arvind A., Shivnath B., and Jennifer W. The CQL continuous query language: semantic foundations and query execution. VLDB J., 15(2):121–142, 2006.
4. Bai Y., Thakkar H., Luo C., Wang H., and Zaniolo C. A data stream language and system designed for power and extensibility. In Proc. Int. Conf. on Information and Knowledge Management, 2006, pp. 337–346.
5. Chandrasekaran S. and Franklin M. Streaming queries over streaming data. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 203–214.
6. Cherniack M. SQuAl: The Aurora [S]tream [Qu]ery [Al]gebra, Technical Reprt, Brandeis University, 2003.
7. Codehaus.org, Esper online documentation set, <http://esper.codehaus.org/tutorials/tutorials.html>, 2007.
8. Conway N. An introduction to data stream query processing. Slides from a talk given on May 24, 2007, http://www.pgcon.org/2007/schedule/attachments/17-stream_intro.pdf, 2007.
9. Coral8 Systems, Coral8 CCL Reference Version 5.1, <http://www.coral8.com/system/files/assets/pdf/current/Coral8CclReference.pdf>, 2007.
10. Jennifer W. CQL: a language for continuous queries over streams and relations. Slides from a talk given at the Database Programming Language (DBPL) Workshop, Potsdam, Germany, 2003. <http://www-db.stanford.edu/~widom/cql-talk.pdf>
11. Streambase Systems, StreamSQL online documentation set, <http://streambase.com/developers/docs/latest/streamsql/index.html>, 2007.
12. Wu E., Diao Y., and Rizvi S. High-performance complex event processing over streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 407–418.
13. Zaniolo C., Luo C., Wang H., Bai Y., and Thakkar H. An Introduction to the Expressive Stream Language (ESL), Technical Report, UCLA.

transparent illusion of an unreplicated database. Such a system is described as offering a strong consistency model. 1-copy-serializability (q.v.) is the best-known strong consistency model.

Historical Background

Early work in the 1970s investigated a range of replica control mechanisms, usually with the intention of providing transparent serializability. In the early 1980s, Bernstein and colleagues formalized the concept of 1-copy-serializability as a consistency model [1], with a careful proof technique [2] like that for single-site serializability. Herlihy [8] extended these ideas to replicating data types with general operations (not just read and write).

1996 marked the seminal paper by Gray et al. who used some simple performance models to show the scalability barriers for different system designs [7]. This inspired research on ways to gain 1-copy serializability within a lazy single-master replica control, through restrictions on the placement of copies, and/or the ordering of update propagation [3,4].

Since 2000, the prevalence of DBMS platforms with Snapshot Isolation concurrency control led to research on replication for these, especially by groups of researchers centered on Alonso and Kemme. The concept of 1-copy-SI was defined [10], and a proof theory was developed [9]. Variant consistency models were defined by considering different session properties [5,6].

Foundations

A major theme in the development of distributed databases has been *transparency*, that is, the clients should not have to change if they interact with a distributed system rather than a traditional, single-site database. Transparency applies to many aspects: table naming should be the same as for a single DBMS, queries should not need rewriting even if data is fragmented between sites, etc. The shift from a single-site system to a replicated one should be seen in better quality of service, but not in altered functionality. Since a single-site database has only one copy of each data item, a transparently replicated system will provide clients with the illusion of a single copy, hiding all evidence of the replication. A scheme for replica control (q.v.) that does this can be described as providing a strong consistency model. Other replica control mechanisms do not hide the fact of replication.

There are in fact several variants among strong consistency models, because there are several different

Strong Consistency Models for Replicated Data

ALAN FEKETE

University of Sydney, Sydney, NSW, Australia

Synonyms

[Strong memory consistency](#); [Copy transparency](#)

Definition

If a distributed database system keeps several copies or replicas for a data item, at different sites, then a replica control protocol determines how the replicas are accessed. Some replica control protocols ensure that clients never become aware that the data are replicated. In other words, the system provides the

isolation models used by different DBMS platforms, and because the formal definition of isolation doesn't always capture exactly the properties of an implementation. The next paragraphs describe the main strong consistency models that have been proposed for replicated data.

Replicated Serializability. The theory of concurrency control has an established notion of correct functionality for ACID transactions in a single-site DBMS: serializability (q.v.). This is defined by having execution equivalent to a serial (i.e., batch, non-overlapped) execution of the same transactions. When a replicated database gives to clients the transparent appearance of a single-site system with serializable transactions, one says that the replica consistency model is 1-copy-serializability (q.v.). That is, the operations of the transactions are indistinguishable from what happens if they are run serially in a database with only one site.

To illustrate the consistency model, consider a database with two logical items, x and y representing respectively the balance in the checking and savings bank accounts for a single customer. There is a single client C which submits two transactions. One client transaction $T_{1,C}$ is a transfer of two units of funds from y to x , and $T_{2,C}$ is a transaction to display the status of the customer's finances. Initial values are $x=10$ and $y=20$.

In (1) below is a sequence of events that might happen in a system with two sites A and B , using an eager locking-based read-one-write-all replica control. Notice how each client-submitted transaction has sub-transactions at the local sites A and B ; $T_{1,C}$ reads both items at site A , and then updates replicas at both sites; while $T_{2,C}$ reads the replica x^A and the replica y^B . The notation that is used in this and later examples is to indicate the event where a value 5 is written to the local replica of item x at site A , as part of transaction T_1 , by $w_1[x^A, 5]$. Here the subscript on the event type indicates the transaction involved, and the superscript on the item name indicates the site of the replica which is affected. The event where a client C running transaction T_3 has requested a read of the logical data item y , and the value 6 is returned, will be represented by $r_{3,C}[y, 6]$. Note there is no superscript on the item since this is the client's view. Many consistency models need to refer to where transactions start and finish, so one also has events like $b_{3,C}$ for the start of transaction T_3 at client C , or $c_{3,C}$ for the commit of that transaction by the client, or indeed c_3^A for the commit of the local subtransaction of T_3 which is running at site A .

$$\begin{aligned} & b_{1,C} b_1^A r_1[x^A, 10] r_{1,C}[x, 10] b_{2,C} b_2^A r_1[y^A, 20] \\ & r_{1,C}[y, 20] w_1[x^A, 12] b_1^B w_1[x^B, 12] w_{1,C}[x, 12] \\ & w_1[y^A, 18] w_1[y^B, 18] w_{1,C}[y, 18] c_1^A r_2[x^A, 12] \\ & r_{2,C}[x, 12] b_2^B c_1^B c_{1,C} r_2[y^B, 18] \\ & r_{2,C}[y, 18] c_2^A c_2^B c_{2,C} \end{aligned} \quad (1)$$

When one hides the internal details (the events at the replicas), and only considers what the client sees, the relevant event sequence is

$$\begin{aligned} & b_{1,C} r_{1,C}[x, 10] b_{2,C} r_{1,C}[y, 20] w_{1,C}[x, 12] \\ & w_{1,C}[y, 18] r_{2,C}[x, 12] c_{1,C} r_{2,C}[y, 18] c_{2,C} \end{aligned} \quad (2)$$

This sequence is 1-copy-serializable, because the client sees the same as in a serial execution on a single-site DBMS, in the order $T_{1,C}$ then $T_{2,C}$.

Here is a sequence that might occur in a system where site A is the primary or master site, where all updates are initially done, and site B has secondary replicas which are updated through copier transactions that lazily apply the write sets of any update transaction. The copier that transmits values produced by $T_{1,C}$ from site A to site B will be denoted by T_{\oplus}^B ; notice that in contrast to T_1^B in the eager system modeled previously, T_{\oplus}^B is not a subtransaction of any global client-submitted transaction but instead it can commit independently.

$$\begin{aligned} & b_{1,C} b_1^A r_1[x^A, 10] r_{1,C}[x, 10] r_1[y^A, 20] r_{1,C}[y, 20] \\ & w_1[x^A, 12] w_{1,C}[x, 12] w_1[y^A, 18] \\ & w_{1,C}[y, 18] c_1^A c_{1,C} b_{2,C} \\ & b_2^B r_2[x^B, 10] r_{2,C}[x, 10] r_2[y^B, 20] r_{2,C}[y, 20] c_2^B \\ & c_{2,C} b_{\oplus}^B w_{\oplus}[x^B, 12] w_{\oplus}[y^B, 18] c_{\oplus}^B \end{aligned} \quad (3)$$

Again hiding the internal details, and considering only what the client sees, it is

$$\begin{aligned} & b_{1,C} r_{1,C}[x, 10] r_{1,C}[y, 20] w_{1,C}[x, 12] \\ & w_{1,C}[y, 18] c_{1,C} \\ & b_{2,C} r_{2,C}[x, 10] r_{2,C}[y, 20] c_{2,C} \end{aligned} \quad (4)$$

This sequence is also 1-copy-serializable, since the values read are what could happen with an unreplicated system running the transactions serially in the order $T_{2,C}$ then $T_{1,C}$.

There is a detailed theory that allows one to prove that this property holds for schedules of certain read-one-write-all replica control mechanisms.

Session properties. The sequence of events shown in sequence (4) is indeed something that could happen

according to the definition of serializable execution in a single-site DBMS, but it would never happen in a single site DBMS that used a concurrency control mechanism like two-phase locking. It might be very disturbing to be a client, who submits a transfer transaction $T_{1,C}$, learns that the transfer succeeded, and then uses $T_{2,C}$ to check the status of their accounts and is told that the initial balances are still unchanged. In the definition of serializability, it is enough that there exists some way to order the transactions and perform them serially; but in DBMS products, the concurrency control makes sure that the apparent serial order does not rearrange transactions unless they are actually concurrent (that is, unless they overlap). Any single-site system will not allow a situation where $T_{1,C}$ has completed, and then $T_{2,C}$ starts, but the apparent serial order has an inversion of the transaction order, with $T_{2,C}$ coming first. Thus it is often proposed to have explicit session properties [11] in a consistency model, to require that the apparent order has some relationship to what really happened. A very restrictive session requirement is *external consistency*; this means that whenever $T_{i,C}$ completes before $T_{j,D}$ starts, then the apparent serial order must contain $T_{i,C}$ ahead of $T_{j,D}$. A less restrictive property is *session consistency*, which says that the apparent order must not rearrange transactions from the same client, where one completes before the other starts. That is, whenever $T_{i,C}$ completes before $T_{k,C}$ starts, then the apparent serial order must contain $T_{i,C}$ ahead of $T_{k,C}$. But session consistency says nothing about the serialization order of $T_{i,C}$ and $T_{j,D}$ where $C \neq D$.

Replicated Snapshot Isolation. Several prominent single-site DBMS platforms provide isolation for transactions using a multiversion mechanism called Snapshot Isolation. This does not have exactly the properties of Serializability, but it avoids most of the known bad concurrency problems, and it seems to satisfy most application programmers. The key to this isolation level is that when a transaction reads an item, it sees the value which reflects all writes by other transactions which committed before the reading transaction started, but it does not see any effects of concurrent transactions.

When replicating data stored in platforms that offer Snapshot Isolation, a natural strong consistency model is to transparently appear like an unreplicated system running on the same sort of platform. This is

the consistency model known as “1-copy-SI” [10]. Here is an example with concurrent client transactions $T_{3,C}$ and $T_{4,D}$, each of which reads two logical data items and increments one of them. The sequence (5) is something that a client could see in 1-copy-SI, but not in a system offering 1-copy-serializable consistency, because the values read are not the same as the serial order $T_{3,C}$ then $T_{4,D}$ (where $T_{4,D}$ would read $y = 21$), nor are they as in $T_{4,D}$ followed by $T_{3,C}$ (where $T_{3,C}$ would read $x = 11$).

$$\begin{aligned} b_{4,D} r_{4,D}[x, 10] & r_{4,D}[y, 20] b_{3,C} r_{3,C}[x, 10] \\ r_{3,C}[y, 20] w_{4,D}[x, 11] & w_{3,C}[y, 21] c_{3,C} c_{4,D} \end{aligned} \quad (5)$$

As the definition of Snapshot Isolation says that a read sees the effects of all transactions that commit before the reader’s transaction started, this definition implicitly includes an external consistency session property. To allow more efficient replica control algorithms, some researchers have built systems which provide more permissive consistency models. For example, in Generalized Snapshot Isolation [6], for each transaction there is a snapshot-time, which could be somewhat before the transaction starts, and the transaction’s reads see exactly the effects of those transactions that committed before the reader’s snapshot-time. This allows the strange inversions where a client submits a transaction, learns of its success, and then submits another transaction that does not run in the expected state. Thus, an intermediate consistency model is Strong Session Snapshot Isolation [5], where inversions are allowed between non-concurrent transactions from different clients, but the snapshot-time for a transaction must be later than the commit of any previous transaction submitted by the same client.

Key Applications

1-copy serializability is most often provided through eager or synchronous propagation of updates among machines that are all in a single cluster. Some database engines provide options for replication internally, whereas others rely on replication tools that run along side the DBMS engine. So far, the model of 1-copy serializability with lazy propagation of updates, or the model of replicated snapshot isolation, are offered in research prototypes rather than among commercial products.

Future Directions

The algorithms known for replicated snapshot isolation may be attractive for practical use, since they can offer substantially higher performance than the algorithms for 1-copy serializability, and since programmers seem able to work with snapshot isolation in a single DBMS. Thus, this strong consistency model will probably become more widespread in the future.

Cross-references

- ▶ [Data Replication](#)

Recommended Reading

1. Attar R., Bernstein P.A., and Goodman N. Site initialization, recovery, and backup in a distributed database system. *IEEE Trans. Software Eng.*, 10(6):645–650, 1984.
2. Bernstein P.A. and Goodman N. Serializability theory for replicated databases. *J. Comput. Syst. Sci.*, 31(3):355–374, 1985.
3. Breitbart Y., Komondoor R., Rastogi R., Seshadri S., and Silberschatz A. Update propagation protocols for replicated databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
4. Chundi P., Rosenkrantz D.J., and Ravi S.S. Deferred updates and data placement in distributed databases. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 469–476.
5. Daudjee K. and Salem K. Lazy database replication with snapshot isolation. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 715–726.
6. Elnikety S., Zwaenepoel W., and Pedone F. Database replication using generalized snapshot isolation. In Proc. 22nd Symp. on Reliable Distributed Syst., 2005, pp. 73–84.
7. Gray J., Helland P., O’Neil P.E., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
8. Herlihy M. A quorum-consensus replication method for abstract data types. *ACM Trans. Comput. Syst.*, 4(1):32–53, 1986.
9. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based data replication providing snapshot isolation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 419–430.
10. Plattner C. and Alonso G. Ganymed: Scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004, pp. 155–174.
11. Terry D.B., Demers A.J., Petersen K., Spreitzer M., Theimer M., and Welch B.B. Session guarantees for weakly consistent replicated data. In Proc. Int. Conf. on Parallel and Distributed Information Systems, 1994, pp. 140–149.

Strong Coupling

- ▶ [Tight Coupling](#)

Strong Memory Consistency

- ▶ [Strong Consistency Models for Replicated Data](#)

Structural Index

- ▶ [Structure Indexing](#)

Structural Summary

- ▶ [Structure Indexing](#)

Structure Indexing

MARIANO P. CONSENS

University of Toronto, Toronto, ON, Canada

Synonyms

[Structural index](#); [Structural summary](#); [Path index](#); [Dataguide](#); [Synopsis](#); [Sketch](#)

Definition

Structure indexing creates summaries of the structure present in semi-structured data collections by grouping data items with similar structure, providing a mechanism to index such items. Since semi-structured data models are commonly represented by labeled graphs or trees (the XML data model being a prime example), structural indexes or summaries are naturally described as graphs where nodes represent sets of data items (called extents), and where edges represent structural relationships between the corresponding extents derived from the instance data. A concrete physical index can be created by selecting appropriate data structures to store the graph and the extents.

Structure indexing helps to find data items that satisfy structural constraints in queries by locating nodes in the structural summary graph that satisfy the query conditions (expecting far less summary nodes than data items), and then limiting query evaluation to data items in the relevant extents.

Structural summaries also provide a description of the structure present in the instance. This is in contrast

with schemas, which prescribe structures that may or may not occur in an instance, but without giving an indication of the metadata that is actually present in a given collection. Note that it is possible to create a structural summary from an instance even when the instance does not conform to any schema.

Additional information can be attached to summaries (such as statistics related to nodes and relationships, and distributions of values associated with the extents), with applications in selectivity estimation and query optimization.

Historical Background

Structure indexing mechanisms were proposed as soon as the database community turned its attention to the problem of managing semi-structured data. Region Inclusion Graphs (RIG) and Region Order Graphs (ROG) [3] were proposed to help optimize the evaluation of region algebras (see [14]).

Representative Objects (RO) [10] and Dataguides [6] were motivated by a desire to describe the metadata present in semi-structured databases (modeled as labeled graphs), as well as to help in query optimization. The representative object of length 1 (1-RO) coincides with a RIG; both summaries group data items in the instance (nodes or regions, respectively) based on the label of an item. Representative objects of length k (k -RO) group instance nodes by the labels in the incoming paths of length k (or paths of arbitrary length in the case of a full representative object, or FRO). Dataguides create a summary of the path structure of a labeled graph database instance in which every label path starting at the root appears exactly once. Construction of a Dataguide is analogous to the conversion of a non-deterministic finite automaton (describing the language of the labels occurring in the instance) to a deterministic one. This construction is not unique, but a strong Dataguide is defined to be suitable as an index. For arbitrary graph data, Dataguides can in the worse case become exponentially larger than the actual instance. When instances are trees, as is the case for XML documents when links (such as IDREF) are not considered, the Dataguide size is bounded by the size of the instance.

The concept of bisimulation was introduced in the context of structural summaries by the T-index family [9]. In particular, the 1-index defines extents via partitioning the nodes of the instance using labeled

bisimulation on the incoming paths. By creating a partition, bisimulation based summaries have a size bounded by the size of the instance. The F&B-Index [7] generalized the notion of bisimulation-based summaries to consider partitions created by both incoming and outgoing paths. AxPRE summaries [4] introduced a language for defining the neighborhood of interest to the bisimulation-based partitioning criteria. Depending on the axis path regular expression (AxPRE) used, all of the previously proposed bisimulation-based summaries can be defined (as well as entirely new ones).

There are also proposals that augment structural summaries with statistical information of the instance for selectivity estimation, including path/branching distribution and value distributions (e.g., XSketch [11], StatiX [5]).

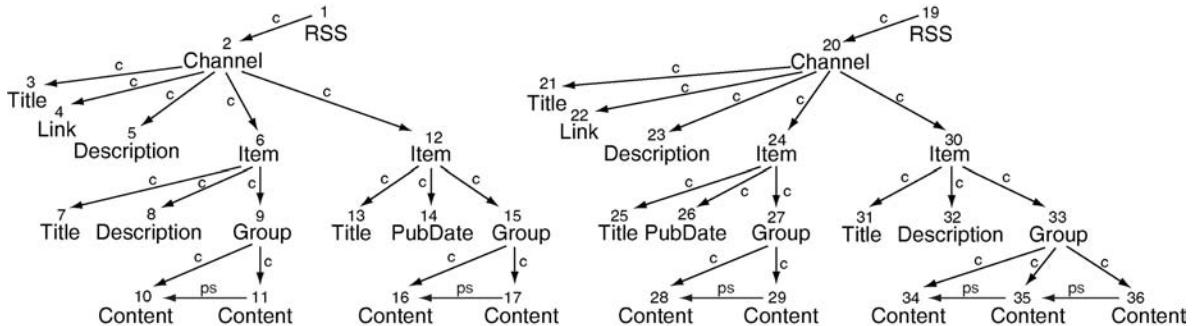
Most of the existing summary proposals define all the extents using the same criteria, hence creating homogeneous summaries. These summaries are based on common element paths (in some cases limited to length k), including incoming paths (e.g., representative objects [10], dataguides [6], 1-index [9], ToXin [13], A(k)-index [8]), both incoming and outgoing paths (e.g., F&B-Index [7]), or sequences of outgoing paths (e.g., Skeleton [1]). There are also heterogeneous summaries where summary nodes adapt their criteria to query workloads (APEX [2], D(k)-index [12]), or to statistics from the instance (XSketch [11]), or where the criteria can be given explicitly (AxPRE summaries [4]).

Foundations

A suitable graph-based model for semi-structured instances is introduced below and then partition-based structural summaries are defined. The discussion then generalizes this definition to AxPRE summaries and then presents a lattice with several bisimulation-based summaries in the literature.

Semi-structured Data Example

Consider an example XML instance that consists of two RSS documents, represented as two labeled graphs in Fig. 1. RSS documents are used to encode feeds that publish frequently updated Web content such as news headlines, blog entries, and podcasts. The feeds are organized into *channels*, which in turn contain a list of *items*. A number of elements (such as *title*, *link*, *description*, *pubDate*) can optionally appear within channels and/or items. The items may have *content*,



Structure Indexing. Figure 1. Sample XML instance with two RSS feeds.

appearing within *groups*, that refers to multimedia files (e.g., an audio file in a podcast). The labels in the XML nodes (which are numbered for ease of reference) identify the element, and the labels in the edges the relationship (or *axis* in the XPath data model) between the XML nodes. The edge from node 6 to 7 labeled *c* means that 7 is the child of 6 (or, considering the inverse relationship, that 6 is the parent of 7). Similarly, the edge from node 29–28 labeled *ps* means that 28 is the preceding sibling of 29 (another XPath axis that provides information about the XML document order among siblings).

Axis Graph Definition

An *axis* graph is defined to represent XML instances. The definition can be easily applied to other semi-structure data models where instances are represented by labeled graphs.

An axis graph $\mathcal{A} = (\text{Inst}, \text{Axes}, \text{Label}, \lambda)$ is a structure where Inst is a finite set of nodes, Axes is a set of binary relations $\{E_1, \dots, E_n\}$ in $\text{Inst} \times \text{Inst}$, Label is a finite set of node names, and λ is a function that assigns labels in Label to nodes in Inst . Edges in \mathcal{A} are labeled by the name of the axes relations.

Structural Summary Definition

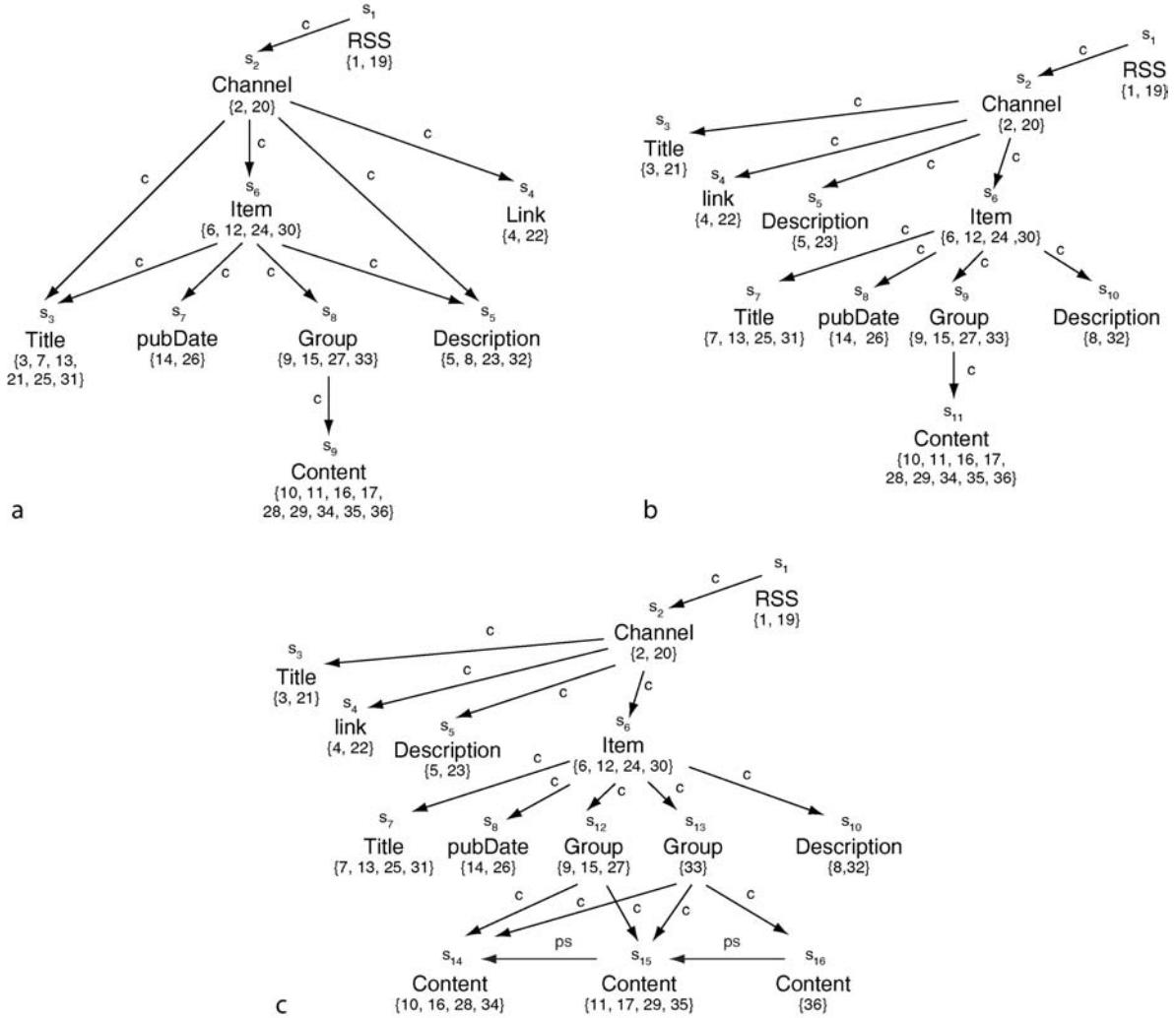
A structural summary of an axis graph \mathcal{A} is another axis graph $\mathcal{S}(\mathcal{A}) = (\text{Sum}, \text{Axes}_{\mathcal{S}}, \text{Label}, \lambda_{\mathcal{S}})$ where summary nodes correspond to a partition of the nodes in \mathcal{A} and summary edges are induced from the axes in \mathcal{A} . Each node $s \in \text{Sum}$ with $\lambda_{\mathcal{S}}(s) = l$ corresponds to a subset (called extent) in a partitioning of Inst , such that all the nodes in the extent have the same label $l \in \text{Label}$. Also, for every edge $E_i(n, m)$ with $n, m \in \text{Inst}$ and $E_i \in \text{Axes}$ there is $E'_i(n_s, m_s)$ with n_s ,

$m_s \in \text{Sum}$ and $E'_i \in \text{Axes}_{\mathcal{S}}$, such that n_s (respectively, m_s) is the summary node that has XML node n (respectively, m) in its extent.

Sample Structural Summaries Figure 2 shows three different structural summaries of the two RSS documents depicted in Fig. 1. The first summary, in Fig. 2a, is constructed by a partition of the XML instance nodes based solely on their labels, and as such there is a single summary node for each label in the instance. As such, the extent of the summary node s_9 labeled *description* consists of the subset $\{5, 8, 23, 32\}$ of all the XML nodes labeled *description* in the instance. Note that there are two edges labeled *c* incoming into the summary node s_9 since *description* appears both in *channels* and *items*.

The second summary, in Fig. 2b, is constructed from a partition that groups together XML nodes with the same ancestor elements. Note that there are two summary nodes (s_5 and s_{10}) labeled *description*, corresponding the two possible paths of labels to the root (*channel*, *RSS*, and *item*, *channel*, *RSS*). Observe that, for the specific instance in the example, grouping together nodes with just the same parents would produce the same summary. However, grouping together nodes with the same children would produce a different summary (e.g., there would be two summary nodes for *item*, since the *item* nodes in the XML instance would be partitioned into the sets $\{6, 30\}$ and $\{12, 24\}$ each with the same sets of children).

The third summary, in Fig. 2c, contains an heterogeneous summary, where subsets in the partition group XML nodes with different criteria. While most the summary nodes in the figure have extents containing XML nodes that use the same criteria as above (having the same ancestor elements), the summary nodes corresponding to *content* elements are



Structure Indexing. Figure 2. Summaries of the two RSS feeds; (a) label, (b) ancestors, (c) heterogeneous.

partitioned according to the previous siblings (the edges labeled *ps* in the instance) and the *group* elements are partitioned according to the partition of their *content* children. So there are three summary nodes s_{14}, s_{15} , and s_{16} that group the first, second, and third *content* node siblings in the instance, and there are two summary nodes s_{12} and s_{13} corresponding, respectively, to those *group* elements that have a first and second *content* node as children, or those that have a first, second, and third *content* node as children.

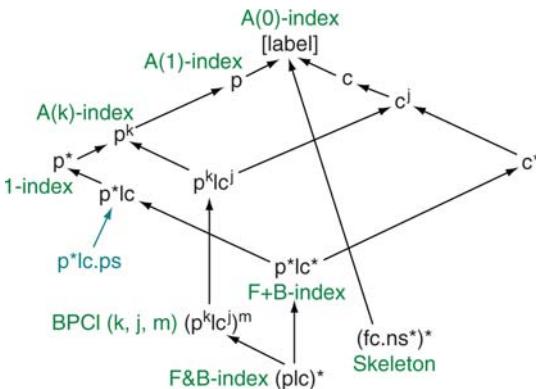
AxPRE Summary Definition

An axis path regular expression (AxPRE) is a regular expression on the vocabulary of the names of the axes

relations. The AxPRE Neighborhood $\mathcal{N}_\alpha(v)$ of an instance node $v \in Inst$ is the subgraph obtained by intersection with the prefix closed finite automaton corresponding to the AxPRE α .

A *labeled bisimulation* between two subgraphs \mathcal{G}_1 and \mathcal{G}_2 of an axis graph \mathcal{A} is a symmetric relation \approx such that for all $v \in Inst^{\mathcal{G}_1}$, $w \in Inst^{\mathcal{G}_2}$, $E_i^{\mathcal{G}_1} \in Axes^{\mathcal{G}_1}$, and $E_i^{\mathcal{G}_2} \in Axes^{\mathcal{G}_2}$: (i) if $v \approx w$, then $\lambda(v) = \lambda(w)$; (ii) if $v \approx w$, and $\langle v, v' \rangle \in E_i^{\mathcal{G}_1}$, then $\langle w, w' \rangle \in E_i^{\mathcal{G}_2}$ and $v' \approx w'$.

An AxPRE summary is a structural summary where the partition is defined as follows: two nodes $v, w \in Inst$ belong to the same partition block iff there exists a *labeled bisimulation* \approx between $\mathcal{N}_\alpha(v)$ and $\mathcal{N}_\alpha(w)$ such that $v \approx w$.



Structure Indexing. Figure 3. Summary lattice.

Sample AxPRE Summaries

The summary in Fig. 2a corresponds to the AxPRE summary with an empty AxPRE, where only condition (i) in the definition of labeled bisimulation applies and the result is a partition of the instance nodes according to their labels only.

The summary in Fig. 2b corresponds to the AxPRE summary with an AxPRE p^* , which is the AxPRE that creates node neighborhoods that consists of all the ancestors of the node. As discussed earlier, for the specific instance in the example, the AxPRE could also be p , but it can not be c .

The summary in Fig. 2c is an heterogeneous summary where most summary nodes have the partition defined according to the AxPRE p^* , but the summary nodes labeled *content* use $p^*|ps^*$ and those labeled *group* use $p^*|c.ps^*$.

Summary Lattice

Figure 3 shows a lattice with relationships among several AxPRE summaries that capture bisimilarity-based proposals mentioned earlier. The lattice represents the partition refinement relationship between the summaries. Each node in the lattice of Fig. 3 corresponds to a homogeneous summary defined by the AxPRE label, with an additional textual label for the corresponding summary name. A node is also used to represent an homogeneous AxPRE summary based on the AxPRE $p^*|c.ps^*$, which applied to only some of the summary nodes in the example in Fig. 2c.

Key Applications

Indexing, metadata description, query processing and optimization, selectivity estimation.

Future Directions

Structure indexing techniques can be extended to support additional data models and their associated query languages, and to further refine their adaptability to different workloads.

Cross-references

- Bisimulation
- Indexing
- Query Processing and Optimization in Object Relational Databases
- Selectivity Estimation
- Semi-structured Data
- Statistical Summaries
- XML
- XPath Data Model
- XPath/XQuery

Recommended Reading

1. Buneman P., Choi B., Fan W., Hutchison R., Mann R., and Viglas S. Vectorizing and querying large XML repositories. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 261–272.
2. Chung C.-W., Min J.-K., and Shim K. APEX: an adaptive path index for XML data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 121–132.
3. Consens M.P. and Milo T. Optimizing queries on files. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994, pp. 301–312.
4. Consens M.P., Rizzolo F., and Vaisman A.A. AxPRE summaries: exploring the (semi-)structure of XML web collections. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 1519–1521.
5. Freire J., Haritsa J.R., Ramanath M., Roy P., and Simeon J. StatiX: making XML count. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 181–191.
6. Goldman R. and Widom J. Dataguides: enabling query formulation and optimization in semistructured databases. In Proc. 23rd Int. Conf. on Very Large Data Bases, 1997, pp. 436–445.
7. Kaushik R., Bohannon P., Naughton J.F., and Korth H.F. Covering indexes for branching path queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 133–144.
8. Kaushik R., Shenoy P., Bohannon P., and Gudes E. Exploiting local similarity for indexing paths in graph-structured data. In Proc. 18th Int. Conf. on Data Engineering, 2002, pp. 129–140.
9. Milo T. and Suciu D. Index structures for path expressions. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 277–295.
10. Nestorov S., Ullman J.D., Wiener J.L., and Chawathe S.S. Representative objects: concise representations of semistructured, hierarchical data. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 79–90.
11. Polyzotis N. and Garofalakis M.N. XSketch synopses for XML data graphs. ACM Trans. Database Syst., 31(3):1014–1063, 2006.

12. Qun C., Lim A., and Ong K.W. D(k)-index: an adaptive structural summary for graph-structured data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 134–144.
13. Rizzolo F. and Mendelzon A.O. Indexing XML data with ToXin. In Proc. of Fourth Int. Workshop on the Web and Databases, 2001, pp. 49–54.
14. Young-Lai M. and Tompa F.W. One-pass evaluation of region algebra expressions. *Inform. Syst.*, 28(3):159–168, 2003.

Using binary weights means that an element is (value one) or is not (value zero) considered for indexing and retrieval. The decision can be made by looking at the DTD (document type definition) of the collection, past relevance data, and/or the requirements of the application and user scenario. In the selective indexing strategy [3], only elements of types that were found to contain relevant content for previous query sets (relevance data) are considered. Any elements with a length size less than a given threshold can also be ignored.

Weights can be assigned to characteristics of elements, such as length, depth, location in the document logical structure, and so on. For instance, within the language modelling framework, length has been used as a normalization parameter (weight) incorporated through a prior probability in the ranking formula [2].

With statistical approaches, the weights are estimated based on training data, such as past relevance data. The weights can be determined using machine learning, and then used in the ranking function. They can also be directly calculated based on the distribution of element characteristics. For example, in [1], the distribution of tag types is used in a way similar to the binary independence retrieval model (investigating the “presence” of tags in relevant and non-relevant elements) to estimate the element weights.

Structure of Truth Values

- Residuated Lattice

Structure Weight

MOUNIA LALMAS

University of London, London, UK

Definition

In structured text retrieval, the structure of a text component may be used to estimate the relevance of that component. This is done by associating a weight to the structure reflecting its significance when estimating the relevance of the component for a given query.

Key Points

Associating weight to the structure of a component in itself is not new, and several investigations have been reported for whole document retrieval. This entry is concerned with structure weights in the context of structured text retrieval, where the aim is to exploit the document structure to return document components, instead of whole documents.

In structured text retrieval, not all document components will trigger the same user satisfaction when returned as answers to queries. In the context of structured documents mark-up in XML, some document components, i.e., XML elements, may not be appropriate to return because they are too small, or a tag type that does not contain informative content, nested too deep in the document logical structure, or for other reasons. When ranking XML elements, their structure (size, tag type, path, depth, etc.) may prove important. The importance of the element structure is captured through a weight, which can be binary.

Cross-references

- Indexing Units
- Logical Structure
- Relationships in Structured Text Retrieval
- XML Retrieval

Recommended Reading

1. Gery M., Largeron C., and Thollard F. Probabilistic document model integrating XML structure. In Proc. 6th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2007, pp. 139–149.
2. Kamps J., de Rijke M., and Sigurbjörnsson B. Length normalization in XML retrieval. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 80–87.
3. Mass Y. and Mandelbrod M. Component ranking and automatic query refinement for XML retrieval. In *Advances in XML Information Retrieval*, Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, Revised Selected Papers, 2005, pp. 73–84.

Structured Data in Peer-to-Peer Systems

KAI-UWE SATTLER

Technical University of Ilmenau, Ilmenau, Germany

Synonyms

Peer data management; Peer database management; P2P database

Definition

A peer database management system is a peer-to-peer (P2P) system that manages structured data. Each node in such a system maintains data which conforms to user- or application-defined structures and can be accessed and retrieved efficiently. Examples of data structures are relations as set of tuples and hierarchical or tree-organized data such as XML data and documents. In contrast to unstructured data (e.g., text and binary objects) structured data can be retrieved by specifying logical conditions and further processed by operations such as set operations, aggregations and joins.

Historical Background

One of the origins of coordinator-free distributed data management were scalable distributed data structures (SDDS). One of these approaches called LH* [7] is an extension of linear hashing to distributed nodes, and supports key search as well as parallel operations like hash joins and scans.

Distributed hash-tables (DHT) [2] for managing key/object pairs in very large and changing networks were introduced around 2001, e.g., Chord, CAN or P-Grid. Based on these DHTs several approaches for supporting more database-like functionality (e.g., multi-attribute queries, join and aggregation queries) have been developed. One of the first systems was PIER [5], other examples are RDFPeers [3] as well as the work by Triantafillou and Pitoura [12].

The idea of exploiting the P2P paradigm for data integration in Peer Data Management Systems (PDMS) was first presented in 2002/2003 by systems like Edutella [8], that uses RDF/RDFS as schema language and an own RDF-based query language, and Piazza [4] which is based on XML and XQuery.

Foundations

A widely accepted classification of P2P systems distinguishes between unstructured and structured systems. In the following, techniques for managing and querying structured data in each of these two classes of systems are described.

Unstructured P2P Systems

In an unstructured P2P system peers do not maintain information about the resources managed by other peers. This means that for answering a query the request has to be forwarded to its neighbors. Despite rather simple approaches with fixed schema (e.g., file-sharing systems) a typical example of an unstructured P2P system is a peer data integration system also called PDMS (Peer Data Management System).

A query in a PDMS can be issued at each peer. The peer processes the query locally but has to forward it (or portions of this query) to the neighbor peers which provide relations relevant to this query. For this purpose, the query is rewritten using the correspondence mappings. Depending on the formalism of mappings this is implemented by query unfolding (in case of GaV) or by techniques for answering queries using views (in case of LaV) known from data integration techniques. As long as the rewritten queries still contain views or subgoals instead of only base relations the query has to be further rewritten by the neighbor peer and to be forwarded to its neighbors. The remaining steps of optimization and execution follow the basic principle of distributed query processing. However, there are several ways for further optimizations, e.g., for query routing by using routing indexes or by exploiting transitivities of the correspondence paths.

Structured P2P Systems

In a structured P2P system each peer maintains information about the resources stored at other peers, e.g., in the form of a routing table. A typical example of a structured P2P system is a distributed hashtable (DHT) – a system where a logical key space is partitioned among all peers in order to manage key-object pairs $\langle k, v \rangle$.

In order to manage structured data comprising more than a key and a value component a mapping between the schema of a database relation or XML document and the $\langle k, v \rangle$ pair is needed. In the following, three possible alternatives are presented.

In the *horizontal* or tuple-based approach, each tuple is identified and inserted into the DHT by a resource identifier (or primary key) *oid*. The remaining attributes A_i with the values v_i are treated as a single object:

$$\langle oid, A_1 : v_1, A_2 : v_2, \dots, A_n : v_n \rangle \Rightarrow \\ \langle h(oid), [A_1 : v_1, A_2 : v_2, \dots, A_n : v_n] \rangle$$

This data organization scheme allows to retrieve tuples by their resource identifier. Note, that the individual attribute values are still available, but cannot be accessed directly using the DHT lookup feature. In this way, the DHT can be seen as an index structure for the database relation.

If efficient access to the other attributes A_i is needed, too, additional indexes have to be constructed by inserting additional pairs $\langle [A_i : v_i], oid \rangle$ into the DHT. Instead of using the *oid* as value it is also possible to store the whole tuple.

Furthermore, this approach can be easily extended to more than one relation by adding a name space or relation prefix to the resource identifier. An example system following the horizontal approach is the PIER system [5] which is based on the Bamboo DHT.

An alternative approach is a *vertical* data organization where each tuple is represented by a set of triples:

$$\langle oid, A_1 : v_1, A_2 : v_2, \dots, A_n : v_n \rangle \Rightarrow \{ \langle h(oid), A_1, v_1 \rangle, \\ \langle h(oid), A_2, v_2 \rangle, \dots, \langle h(oid), A_n, v_n \rangle \}$$

In order to be able to query all attributes each of these triples is inserted into the DHT using the resource identifier, the attribute and eventually the value component as key, i.e., $\langle h(oid), [A_1 : v_1] \rangle$, $\langle h(A_1), oid \rangle$, $\langle h(v_1), oid \rangle$. If the DHT supports prefix search, the concatenation of $A_i \# v_i$ can be used as key to allow queries both for A_i as well as $A_i = v_i$. Advantages of this vertical data organization are first, that triples with similar values are stored at the same peer or at least in the neighborhood which simplifies joins and – in combination with an order-preserving hash function – range queries, and similarity queries. Second, there is no fixed schema for a given relation. Users can extend the schema to their needs by simply adding new triples for a given tuple. The disadvantage is the required storage overhead.

This approach is not restricted to relational data. RDF data which can be represented using triples of

subject, predicate, and object can be directly mapped to this scheme. Examples of systems following this idea are RDFPeers [3] based on MAAN a multidimensional extension of Chord, and UniStore [6] which is built on top of P-Grid.

Assuming a P2P system for storing and querying XML documents instead of relations a third possible approach is to exploit the idea of *path indexing* [11]. A standard technique for indexing XML documents is suffix indexing of XML paths. Given a document identified by a URI and an XML path P in this document consisting of the elements $P = e_1/e_2/\dots/e_n$ the following n suffixes can be derived

$$s_1 = e_1/e_2/e_3/\dots/e_n \\ s_2 = e_2/e_3/\dots/e_n \\ s_i = \dots \\ s_n = e_n$$

Now, based on these subpaths the keys for the DHT are calculated and the following pairs are inserted: $\langle h(s_1), [s_1, URI] \rangle$, $\langle h(s_2), [s_2, URI] \rangle$, etc.

Note, that this requires a DHT supporting prefix search.

An important question affecting the choice of the hash function is how to fragment a relation or document collection. A viable solution has to be found between the two extreme cases:

- Distribute tuples according to their relation identifier, i.e., each relation is completely stored at exactly one peer.
- Distribute tuples according to their resource identifier (*oid*) such that tuples are partitioned among a set of peers. However, in this case tuples of the same relation or with similar values of indexed attributes should be stored at neighboring peers in order to support range and nearest neighbor queries efficiently.

Appropriate fragmentation schemes are based on space-filling curves [1] and order preserving hash functions.

For querying structured data in a DHT both basic processing strategies *data shipping* and *query shipping* can be used. With data shipping the DHT is used only as a data storage: data are retrieved from the responsible peers which are identified by applying the hash functions to the value in the query predicate similarly to an index lookup or index scan in a classical DBMS.

Obviously, this also works for the path indexing approach, where the keys for the lookup operation are calculated from the path expression given in an XPath query. As an example consider a peer asking a path query $A//B/C/D//E$. By computing the longest subpath containing only the child axes (in this case $B/C/D$), the peers responsible for this key region can be identified and contacted for evaluating the remaining query locally.

This approach can be extended to process more complex queries including joins. Based on the retrieved tuples of the first relation, the matching tuples of the second relation are retrieved from the DHT by applying the hash function to the join attribute. However, data shipping becomes inefficient if the query operators are not very selective. In this case, very expensive scans of many peers are needed which is impractical in large-scale systems.

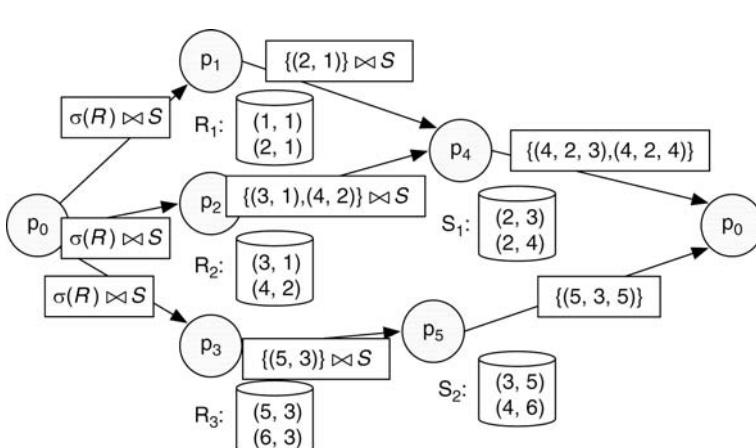
For *query shipping* in a DHT each peer has to provide distributed query processing capabilities rather than only the put/get operations. A query is routed to the peer that is responsible for the data addressed by the operator that is to be executed next. These peers are determined by applying the hash function to the current intermediate results. In this way, the DHT is used both as a hash table for indexing and storing tuples as well as a content-addressable network for routing tuples and/or operators by values.

This idea was exploited in PIER for implementing different join strategies. A first strategy presented in [5] is a variant of the symmetric hash join between two relations R and S . Here, each peer storing tuples of R and S scans its local data and insert these tuples into a temporary table of the DHT. The peers responsible

for the key space of this table execute the probing phase by joining the received tuples locally. A second strategy developed in PIER is a fetch matches variant which can be used if one of the relation is already hashed on the join attribute. In this case, the peers storing tuples of the second relation perform a local scan and retrieve the possibly matching tuples using the get operation of the DHT.

Further improvements can be achieved by applying the idea of symmetric semijoin, i.e., perform a local projection on the join attribute of both relations, followed by a symmetric hash join and feed the results into a fetch matches join to retrieve the remaining attribute values of the original relations. Finally, bloom filters can be used, too. Here, bloom filters are created by each peer responsible for R and S and inserted into temporary DHT table. The received filters are combined and sent to all peers storing the opposite relation. The experiments presented in [5], show that these strategies help to reduce the bandwidth consumption and response time particularly in case of low selectivity values.

In case of more complex queries consisting of sequences of operators query shipping may result in multiple instances of the plan that “travel” through the network, because a single query operator might involve tuples from different peers. This approach, which was initially proposed as *mutant query plans* [9] is illustrated in Fig. 1. Given a relation $R(A, B)$ stored at peers p_1, p_2, p_3 and a relation $S(B, C)$ stored at peers p_4, p_5 . Peer p_0 submits the query $\sigma_{1 < A < 6}(R) \bowtie S$. The query is sent to $p_1 \dots p_3$ which are identified by applying the hash function to the selection



Structured Data in Peer-to-Peer Systems. Figure 1. Mutant query plans in query shipping.

predicate. These peers evaluate the first part of the query ($\sigma_{1 < A < 6}(R)$) locally and replace this expression in the plan by the intermediate result. The modified plan is sent to the S-peers which are identified by applying the hash function on the B values of the intermediate result. Finally, p_4 and p_5 evaluate the remaining parts of the query and send the result back to the initiator.

The benefits of query shipping are exploiting computing resources of the peers as well as avoiding transfer of large datasets. However, query planning and execution is more difficult because the state of a processed query is spread over multiple peers.

Key Applications

A major application for DHT-based P2P database systems is public data management where information of a general interest, its structure and semantics is controlled by a large number of participants. Furthermore, the costs for providing the infrastructure should often be shared by the users in a fair manner. Examples of such applications are the management of public datasets in e-Sciences, e.g., genome data or data in astronomy, metadata and index data for the Semantic Web as well as specialized search engines, naming and directory services as well as social applications such as file/picture sharing, recommender systems or friend-of-a-friend networks. Note, that in these applications it is often not necessary to store the actual data itself in the DHT, but instead metadata or index data required for answering queries are publicly managed.

The main application of unstructured peer data management system is data integration in large-scale, loosely-coupled scenarios.

Future Directions

Managing and querying structured data in P2P systems is a relatively new research area which raises several new challenges to established techniques, e.g., known from distributed database systems.

The first challenge is *scalability*, meaning the support of efficient query processing in networks of ten thousands or more nodes. Most of the research systems presented so far are based on simulation environments or a relatively small number of peers, e.g., in PlanetLab or similar platforms.

A second problem in P2P systems is the dynamic of the network (joining and leaving peers) as well as the

unreliability of peers. Most existing approaches are best effort solutions which are unable to give guarantees wrt. result completeness, freshness or response time. Thus, an open issue is to estimate completeness of results in case of partial answers or to guarantee a certain *quality of service*.

Finally, in large P2P systems where no peer knows all other peers in the network *trustworthiness* are a further challenge. Particularly, if data are redistributed to other nodes in the network the original data producer wants to make sure that its data are not manipulated by the hosting peer. Furthermore, in order to achieve a fair balancing of load and avoiding overloaded peers the problem of rejecting requests and free riding by malicious peers has to be addressed, e.g., by incentive mechanisms.

Cross-references

- ▶ [Distributed Join](#)
- ▶ [Distributed Query Processing](#)
- ▶ [P2P Data Integration](#)
- ▶ [P2P Overlay Networks](#)

Recommended Reading

1. Andrzejak A. and Xu Z. Scalable, Efficient range queries for grid information services. In Proc. of Second IEEE Int. Conf. on Peer to Peer Computing, 2002, pp. 33–40.
2. Balakrishnan H., Kaashoek M.F., Karger D., Morris R., and Stoica I. Looking up Data in P2P Systems. Commun. ACM, 46(2):43–48, 2003.
3. Cai M. and Frank M. RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 650–657.
4. Halevy A., Ives Z., Mork P., and Tatarinov I. Piazza: data management infrastructure for semantic web applications. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 556–567.
5. Huebsch R., Hellerstein J.M., Lanham N., Thau Loo B., Shenker S., and Stoica I. Querying the Internet with PIER. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 321–332.
6. Karnstedt M., Sattler K., Richtarsky M., Müller J., Hauswirth M., Schmidt R., and John R. UniStore: Querying a DHT-based Universal Storage. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 1503–1504.
7. Litwin W., Neimat M.-A., and Schneider D. LH* – a scalable, distributed data structure. ACM Trans. Database Syst., 21(4): 480–525, 1996.
8. Nejdl W., Wolf B., Qu C., Decker S., Sintek M., Naeve A., Nilsson M., Palmer M., and Risch T. Edutella: a P2P networking infrastructure based on RDF. In Proc. 11th Int. World Wide Web Conference, 2002, pp. 604–615.
9. Papadimos V. and Maier D. Mutant query plans. Inform. Software Technol., 44(4):197–206, 2002.

10. Risson J. and Moors T. Survey of Research towards Robust Peer-to-Peer Networks: Search methods. *Comput. Networks*, 50(17): 3485–3521, 2006.
11. Skobeltsyn G., Hauswirth M., and Aberer K. Efficient processing of XPath queries with structured overlay networks. In Proc. Int. Conf. on Cooperative Inf. Syst., 2005, pp. 1243–1260.
12. Triantafillou P. and Pitoura T. Towards a unifying framework for complex query processing over structured peer-to-peer data networks. In Proc. of Int. Workshop on Databases, Information Systems, and Peer-to-Peer Computing, 2003, pp. 169–180.

Structured text retrieval is concerned with the development of models for querying and retrieving from structured text, where the structure is usually encoded with the use of mark-up languages, such as SGML, and now predominantly XML. Indeed, text documents often display structural information. For example, a scientific article will have a so-called logical structure, such as an abstract, several sections, and subsections, each of which is composed of paragraphs. A book will have a so-called layout structure, such as pages and columns.

Structured text retrieval is to be contrasted to traditional text retrieval, where the latter is concerned with the retrieval of unstructured text – so-called “raw text” or “flat text.” The use of the term “structured” in “structured text retrieval” is there to emphasize the interest in the structure. Furthermore, structured text retrieval aims to exploit the available structural information to return text fragments (e.g., XML elements) as opposed to entire text documents.

The term “semi-structured” comes mainly from the database community. Traditional database technologies, such as relational databases, have been concerned with the querying and retrieval of highly structured data (e.g., from a student table, find the names and addresses of those with a grade over 80 in a particular subject). Text documents marked-up, for instance, in XML are made of a mixture of highly structured components (e.g., year, author name) typical of database records, and loosely structured components (e.g., abstract, section). Database technologies are being extended to query and retrieve such loosely structured components, called semi-structured data. Databases that support this kind of data, mainly in the form of text with mark-up, are referred to as semi-structured databases, to emphasize the loose structure of the data and use “querying data” instead of “data retrieval.”

From a terminology point of view, structured text retrieval and querying semi-structured data, in terms of end goals, are the same. The difference comes from the fact that in information retrieval, the structure is added, and in database, the structure is loosened. It should, however, be pointed out that research in information retrieval and databases with respect to accessing structured text (or semi-structured data in the form of text) have been concerned, because of historical reasons, with different aspects of the access process, e.g., ranking in information retrieval versus

Structured Document Retrieval

MOUNIA LALMAS¹, RICARDO BAEZA-YATES²

¹University of London, London, UK

²Yahoo! Research, Barcelona, Spain

Synonyms

Structured text retrieval; Querying semi-structured data; Passage retrieval; XML retrieval; Focused retrieval

Definition

Structured document retrieval is concerned with the retrieval of document fragments. The structure of the document, whether explicitly provided by a mark-up language or derived, is exploited to determine the most relevant document fragments to return as answers to a given query. The identified most relevant document fragments can themselves be used to determine the most relevant documents to return as answers to the given query.

Key Points

The aim of this entry is to clarify different terminologies that have been used to refer to or are strongly related to structured retrieval and semi-structured data.

The term “structured document retrieval,” which was introduced in the early to mid 1990s in the information retrieval community, refers to “passage retrieval” and “structured text retrieval.” In passage retrieval, documents are first decomposed into passages (e.g., fixed-size text-windows of words, fixed discourses such as paragraphs, or topic segments through the application of a topic segmentation algorithm). Passages could themselves be retrieved as answers to a query, or be used to rank documents as answers to the query.

efficiency in databases. Nowadays, there is a convergence trend between the two areas (e.g., [1]).

In the late 1990s, the interest in structured document retrieval grew significantly due to the introduction of XML in 1998, which has now became the de-facto format standard for structured documents (or structured text, semi-structured data). Research on XML retrieval was further boosted with the set-up of INEX in 2002, the Initiative for the Evaluation of XML Retrieval, which allowed researchers to compare and discuss the effectiveness of models specifically developed for XML retrieval [2]. Nowadays, XML retrieval is almost a synonym for structured document retrieval, structured text retrieval, and querying semi-structured data.

Structured document retrieval, passage retrieval, structured text retrieval, querying semi-structured data, XML retrieval, all belong to what has recently been called “focused retrieval” [3]. Focused retrieval is concerned with returning the most focused results to a given query. Such focused results include passages, XML elements, and factoid answers (e.g., London being the capital of the UK).

Cross-references

- ▶ [Document Databases](#)
- ▶ [INitiative for the Evaluation of XML Retrieval](#)
- ▶ [Integrated DB&IR Semi-Structured Text Retrieval](#)
- ▶ [Semi-Structured Data](#)
- ▶ [Structured Text Retrieval Models](#)
- ▶ [XML Retrieval](#)

Recommended Reading

1. Amer-Yahia S., Case P., Rölleke T., Shanmugasundaram J., and Weikum G. In Report on the DB/IR panel at SIGMOD 2005. ACM SIGMOD Rec., 34(4):71–74, 2005.
2. Kazai G., Gövert N., Lalmas M., and Fuhr N. The INEX Evaluation Initiative. In Intelligent Search on XML Data, Applications, Languages, Models, Implementations, and Benchmarks. Springer, New York, NY, 2003, pp. 279–293.
3. Trotman A., Geva S., and Kamps J. Report on the SIGIR 2007 workshop on focused retrieval. SIGIR Forum, 41(2):97–103, 2007.

Structured Text Retrieval

- ▶ [Structured Document Retrieval](#)
- ▶ [XML Retrieval](#)

Structured Text Retrieval Models

DJOERD HIEMSTRA¹, RICARDO BAEZA-YATES²

¹University of Twente, Enschede, The Netherlands

²Yahoo! Research, Barcelona, Spain

Synonyms

[Retrieval Models for Text Databases](#)

Definition

Structured text retrieval models provide a formal definition or mathematical framework for querying semi-structured textual databases. A textual database contains both content and structure. The content is the text itself, and the structure divides the database into separate textual parts and relates those textual parts by some criterion. Often, textual databases can be represented as marked up text, for instance as XML, where the XML elements define the structure on the text content. Retrieval models for textual databases should comprise of three parts: (i) a model of the text, (ii) a model of the structure, and (iii) a query language [4]: The model of the text defines a tokenization into words or other semantic units, as well as stop words, stemming, synonyms, etc. The model of the structure defines parts of the text, typically a contiguous portion of the text called element, region, or segment, which is defined on top of the text model’s word tokens. The query language typically defines a number of operators on content and structure, such as set operators and operators like “containing” and “contained-by” to model relations between content and structure, as well as relations between the structural elements themselves. Using such a query language, the (expert) user can, for instance, formulate requests like “*I want a paragraph discussing formal models near to a table discussing the differences between databases and information retrieval*.” Here, “formal models” and “differences between databases and information retrieval” should match the content that needs to be retrieved from the database, whereas “paragraph” and “table” refer to structural constraints on the units to retrieve. The features, structuring power, and the

Structured Query Language

- ▶ [SQL](#)

expressiveness of the query languages of several models for structured text retrieval are discussed below.

Historical Background

The STAIRS system (Storage and Information Retrieval System), which was developed at IBM in the late 1950's, allowed querying of both content and structure. Much like today's On-line Public Access Catalogues, it was used to store bibliographic data in records with fields such as *keywords* and *title*, providing structured search, but no overlapping or hierarchical structures nor full text search. At the end of the 1980s, researchers at the University of Waterloo in Canada pursued database support for the creation of an electronic version of the Oxford English Dictionary. This resulted in a number of models for querying and manipulating content and hierarchical structure such as the parsed strings model [10], PAT expressions [15], the containment model [5] and generalized concordance lists model [7]. Similar approaches were developed elsewhere, such as the proximal nodes model [13] and the nested region model [11]. The interest in structured text retrieval models has grown since the introduction of XML in 1998, and the emergence of standard data retrieval query languages for XML data. One might argue that the structured text retrieval approaches such as PAT expressions and proximal nodes mentioned above are predecessors of XPath. The success of XML in turn has influenced the work on structured retrieval models: XIRQL was proposed in 2000 [9] as an information retrieval extension of XML query languages (XIRQL is an extension of XQL, a predecessor of XPath). More recently, in 2004, NEXI and XQuery and XPath Full-Text have been proposed as query languages for structured text retrieval, as well as examples of structured text retrieval models for the respective query languages [2,12].

Foundations

There are several models of structured text retrieval. Since there is no consensus on how to structure a textual database, this entry addresses several modeling decisions following the taxonomy presented in [4]. The entry only addresses models for text databases, and not text retrieval models in relational databases as for instance provided by SQL/MM. Due to the success of XML, today XML retrieval is almost a synonym for structured text retrieval, although XML retrieval only addresses the explicit, single hierarchy case below.

Explicit vs. Implicit Structure

Most models use *explicit structure*, i.e., they define unambiguously what parts of the textual database are for instance "sections." These models require the database to be structured explicitly and unambiguously, or using terminology from markup languages: the models require the database to be well-formed. This allows easy modeling of nested regions, and powerful structural relationships such as the direct ancestor relationship (i.e., child and parent axis in XPath). The following query might be used in an explicit structure approach to retrieve sections that contain the word "databases":

```
section CONTAINING "databases"
```

Explicit structure is assumed by amongst others the proximal nodes model [13] and the full match model [2]. In systems that use *implicit structure*, however, structure is not explicitly distinguished from content. In these approaches the database is modeled as a sequence of tokens without distinguishing a word token from a markup token. A structural element should, therefore, be constructed at runtime by looking up the opening markup tokens, the closing tokens, and to return those regions starting with a opening token and ending with a closing token. The query above would then be formulated as [11] (here, the operator "... would be pronounced as "following"):

```
("<section> ... </section>") CONTAINING "databases"
```

So, the section element only exists at querying time. Semantically, the query is not different from a content-only query. For instance the query ("all" .. "equal") CONTAINING "created" retrieves regions that start with the word "all," that end with the word "equal" and that contain the word "created," matching for instance the phrase "all men are created equal." Nested elements, or unbalanced tags are handled differently by several approaches. In the Generalized Concordance Lists (GCL) approach [7], nested sections will not be recognized by the system (instead two partially overlapped sections will be returned). In the nested region algebra approach [11] nested elements are returned properly. The approach is implemented as sgrep (structured grep) (<http://www.cs.helsinki.fi/~jaakkol/sgrep.html>). The GCL approach was recently implemented in a research system called Wumpus (<http://www.wumpus-search.org>).

Static vs. Dynamic Structure

The use of implicit structure also implies the use of *dynamic structure*. A system that uses dynamic structure allows operations that define new elements or regions, i.e., elements or regions that were not previously in the database. In XQuery, this is done by element construction, but in some approaches dynamic structure is a natural consequence of the model. As an early example of dynamic structure consider the following bibliographic entry:

John Doe, "Crime," *Police* 6, 2028.

The entry is explicitly structured by the following grammar that functions as a database schema [10]:

```
entry  := author ',' title ',' journal ',' year ';
author := text ;
title  := ' " text "' ;
journal := text digit+ ;
year   := digit digit digit digit ;
text   := ( letter | '' ) + ;
```

A valid database instance contains data that conforms to the grammar. The instance takes the form of a parsed string or “p-string.” Note that the schema does not distinguish the author’s first name(s) from his surname, but this might be done at query time by introducing a small grammar fragment `NameG` that parses the author strings into given names and surnames:

```
NameG := { name := ( givenname ' ') + surname ;
givenname := letter + ;
surname  := letter + ; }
```

The p-strings model provides a simple query language for adding additional grammar fragments. Suppose the bibliographic entry above is `E`, then the following query returns a p-string containing the author element with given name and surname explicitly identified.

(author in E) reparsed by NameG

The construct might be used to search for all authors with surname “Doe” that wrote a journal paper that mentions “grammar” in the title. The p-strings model uses regular expression matching as a core language primitive, and as such dynamic structure is more easily added than in for instance XQuery or XQuery Full-Text.

Single Hierarchy vs. Multiple Hierarchies

Although some fielded search methods use a flat structure to model text, the approaches considered here assume a hierarchical structure of the text database.

The systems that use implicit structure introduced above assume a single hierarchy. Interestingly, many approaches assume multiple structural hierarchies on the same textual database. Each hierarchy might serve a different purpose. For instance, one hierarchy might represent the logical structure of the text, dividing it in chapters, sections, subsections, etc., whereas a second hierarchy might represent the lay-out structure in columns and pages; and a third layer might represent the results of a part-of-speech tagger, etc. Inside a single hierarchy, the structural elements are either disjoint or nested inside each other, but across hierarchies elements may partially overlap, i.e., a subsection might start half way a page, and end on the following page.

Some approaches relate single views in one query [13]. An interesting approach is suggested by Alink [1], who introduces additional XPath steps (`select-narrow` and `select-wide`) that navigate from one hierarchy to another. For instance, the following XQuery Full-Text-like query fragment navigates from the paragraph elements to another hierarchy with a `Verb` element that contains “killed,” and to a hierarchy with a `person` element that contains “Abraham Lincoln”:

```
$doc//paragraph[./select-narrow::Verb
ftcontains "killed" and ./select-wide::person
ftcontains "Abraham Lincoln"]
```

The need for multiple hierarchies is for instance, addressed in the containment model [5], and the proximal nodes model [13]. In several publications, the hierarchies are called “stand-off annotation” or “offset annotation” to stress that the structural information (or annotations) are modeled separately from the textual data.

Exact Matching vs. Ranking

Many of the early structured text retrieval models do not consider ranked retrieval results, or if they do only as an afterthought, i.e., by ranking the retrieval results using a text-only query disregarding the structural conditions in the query [5]. A simple but powerful way to take the structure of the results into account is to apply a standard information retrieval model to the retrieved content, and then propagate element scores or aggregate term weights based on the text structure. In several of these approaches to ranking, propagation or aggregation is guided by weighting the paths to elements by so-called augmentation weights [9] or interpolation parameters [14], to model for instance that a title element is more likely to contain important

information than a bibliography element. Instead of propagating or aggregating the scores from the leaf nodes, *algebraic* approaches include the ranking functionality inside each operator of the query language [2,12]. Ranking might also include relaxation of the query's structural conditions, for instance by relaxing complex queries step-wise to simpler queries [3].

In 2002, Fuhr and Lalmas [8] organized the first workshop of the Initiative for the Evaluation of XML Retrieval. The goal of INEX is to evaluate the quality of the retrieved results, and as such the quality of the ranking provided by the system taking both content and structure into account. The initiative provides a large testbed, consisting of XML documents, queries and relevance judgments on the data, where the relevance judgments are human judgments that define if an XML element is relevant to the query or not. With XML databases and extensions of XML query languages becoming a de-facto standard for structured text retrieval, ranking is one of the main remaining research challenges.

Key Applications

Systems based on structured text retrieval models can be applied to any problem that involves semi-structured text databases. Key applications of the approaches described in this section include: Managing and searching electronic dictionaries such as the Oxford English Dictionary [10,15], managing and searching electronic journals such as the journals of the IEEE [12,8,6], searching stageplays such as the collected works of William Shakespeare [7], and searching hard drives for digital forensics [1].

Cross-references

- ▶ [Aggregation-based Structured Text Retrieval](#)
- ▶ [Information Retrieval Models](#)
- ▶ [NEXI](#)
- ▶ [Propagation-based Structured Text Retrieval](#)
- ▶ [XPath/XQuery](#)
- ▶ [XQuery Full-Text](#)

Recommended Reading

1. Alink W. XIRAF: an XML information retrieval approach to digital forensics. Master's thesis, University of Twente, 2005.
2. Amer-Yahia S., Botev C., and Shanmugasundaram J. TeXQuery: a full-text search extension to XQuery. In Proc. 12th Int. World Wide Web Conference, 2004.
3. Amer-Yahia S., Lakshmanan L.V.S. and Pandit S. FleXPath: flexible structure and full-text querying for XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.

4. Baeza-Yates R.A. and Navarro G. Integrating contents and structure in text retrieval. ACM SIGMOD Rec., 25(1):67–79, 1996.
5. Burkowski F.J. Retrieval activities in a database consisting of heterogeneous collections of structured text. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 112–124.
6. Carmel D., Maarek Y.S., Mandelbrod M., Mass Y., and Soffer A. Searching XML documents via XML fragments. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 151–158.
7. Clarke C.L.A., Cormack G.V., and Burkowski F.J. An algebra for structured text search and a framework for its implementation. Comput. J., 38:43–56, 1995.
8. Fuhr N., Gövert N., Kazai G., and Lalmas M. (eds.). In Proc. 1st Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2002.
9. Fuhr N. and Grossjohann K. XIRQL: a query language for information retrieval in XML. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 172–180.
10. Gonnet G.H. and Tompa F.W. Mind your grammar: a new approach to modelling text. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 339–346.
11. Jaakkola J. and Kilpeläinen P. Nested text-region algebra. Technical report, University of Helsinki, 1999.
12. Mihajlovic V., Blok H.E., Hiemstra D., and Apers P.M.G. Score region algebra: building a transparent XML-IR database. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 12–19.
13. Navarro G. and Baeza-Yates R.A. Proximal nodes: a model to query document databases by content and structure. ACM Trans. Inf. Syst., 15:400–435, 1997.
14. Ogilvie P. and Callan J. Hierarchical language models for XML component retrieval. In Advances in XML Information Retrieval, Lecture Notes in Computer Science 3493. Springer, 2005, pp. 224–237.
15. Salminen A. and Tompa F.W. PAT expressions: an algebra for text search. In Proc. COMPLEX 92, 1992, pp. 309–332.

Structured Text Retrieval Tasks

- ▶ [Presenting Structured Text Retrieval Results](#)

Subject Spaces

HANS-ARNO JACOBSEN
University of Toronto, Toronto, Ontario, Canada

Definition

Subject spaces are a model to formalize publish/subscribe-style interactions and generalize the publish/subscribe concept. Subject spaces subsume existing

publish/subscribe models, such as the channel-based, the topic-based, the type-based, and the content-based publish/subscribe models. Subject spaces go beyond these models by permitting the treatment of publications and subscriptions symmetrically, extending publications to also include expressive filter predicates, introducing the notion of selective publishing, interpreting publications and subscriptions as either stateless or stateful, and generalizing matching to encompass a wide range of possible matching semantics.

Key Points

The subject spaces model is a unifying formal framework to specify, describe and analyze the publish/subscribe concept. Subject spaces encompass existing publish/subscribe models and allow the modeling of new aspects of publish/subscribe-style interactions. Informally, a *subject space* is the set of values a publisher can publish and a subscriber can subscribe to. The plurality of *spaces* refers to all sets of values, possibly overlapping, that publishers and subscribers publish and subscribe to in one particular instantiation of the model, respectively.

Subject spaces treat publications and subscriptions as symmetric constructs. That is, a publication must not be distinguished from a subscription and both are equally expressive. Thus, publications also contain predicates, such as range predicates, equality predicates, non-equality predicates, and string predicates found only in subscriptions in other models. The implications are that in subject spaces, both publications and subscriptions specify sets of values that can be interpreted as regions in space. Traditionally, only subscriptions specify regions in space, while publications are points in space.

In contrast to the content-based model, in subject spaces publications and subscriptions consist of predicates and attribute-value pairs, both of which are optional allowing one to model the respective counterpart in the content-based model.

Moreover, subject spaces treat publications and subscriptions as stateful entities. A publication, once published, is persisted in its associated subject space until explicitly revoked. Subsequent publish operations are updates to the previously published value. A sequence of publish operations can be interpreted as moving the published value through space. Subscriptions are interpreted in the same fashion.

Subject spaces generalize the matching of publications against subscriptions. Since both contain

predicates and attribute-value pairs, a match between a publication and a subscription exists, if the attribute-value pairs of the publication satisfy the predicates of the subscription and the attribute-value pairs of the subscription satisfy the predicates of the publication. In addition, the interpretation of publications and subscriptions as regions in space enables a generalization of matching to encompass additional matching semantics such as overlap, containment, or closeness of publication and subscription regions in the space.

The symmetric treatment of publications and subscriptions enables a notion of *selective publishing*, whereby a publisher specifies predicates to declaratively specify a subset of potential receivers of published messages. This is in contrast to content-based publish/subscribe, where a publication is delivered to all subscribers with matching subscriptions. The symmetric treatment of publications and subscriptions further extends the conventional interpretation of publication, as mere data points, to ranges, regions in space, or other structures. This increases the expressiveness of the subject spaces model substantially, as opposed to other publish/subscribe models.

The state-persistent nature of publications and subscriptions avoids the problem of redundant notifications. This is because subject spaces distinguish between inserting a publication and updating the previously published value. A match occurs if the publication satisfies the subscription, but not at each update of the published values. Existing content-based publish/subscribe systems cannot model this difference and are therefore susceptible to the redundant notification problem. Subject spaces can model both state-persistent and stateless publish/subscribe. The subject spaces model can be used to formalize and model publish/subscribe-style interactions. Applications of concrete realizations of the subject spaces model are similar to applications of publish/subscribe. The increased expressiveness offered by subject spaces will likely enable new applications. For a more detailed treatment of the subject see for example the work by Leung and Jacobsen [1,2].

Cross-references

- ▶ [Channel-Based Publish/Subscribe](#)
- ▶ [Content-Based Publish/Subscribe](#)
- ▶ [Publish/Subscribe](#)
- ▶ [State-Based Publish/Subscribe](#)
- ▶ [Topic-Based Publish/Subscribe](#)
- ▶ [Type-Based Publish/Subscribe](#)

Recommended Reading

1. Leung H.K.Y. Subject space: a state-persistent model for publish/subscribe systems. In Proc. Conf. of the IBM Centre for Advanced Studies on Collaborative Research, 2002, p. 7.
2. Leung H.K.Y. and Jacobsen H.-A. Efficient matching for state-persistent publish/subscribe systems. In Proc. Conf. of the IBM Centre for Advanced Studies on Collaborative Research, 2003, pp. 182–196.

Subject-based Publish/Subscribe

- ▶ Topic-Based Publish/Subscribe

Subspace Clustering Techniques

PEER KRÖGER, ARTHUR ZIMEK

Ludwig-Maximilians University of Munich, Munich, Germany

Synonyms

[Projected clustering](#); [Oriented clustering](#); [Correlation clustering](#); [Bi-clustering](#); [Co-clustering](#); [Pattern based clustering](#)

Definition

Cluster analysis aims at finding a set of subsets (i.e., a clustering) of a data set. A meaningful clustering reflects a natural grouping of the data. In high-dimensional data, irrelevant attributes and correlated attributes make any natural grouping hardly detectable. Specialized techniques aim at finding clusters in subspaces of a high-dimensional data space.

Historical Background

While different weighting of attributes was in use since clusters were derived by hand, the problem of finding a cluster based on a subset of attributes and a specialized solution was first described in 1972 by Hartigan [8]. However, triggered by modern capabilities of massive acquisition of high-dimensional data in many scientific and economic domains, and the first general approaches to the problem [2–4], research did not focus on the problem until 1998. An illustrative problem description and sketches of some earlier algorithms can be found in [12]. The more special topic of pattern-based clustering

is covered in [11]. A general up-to-date overview is presented in [10].

Foundations

Different Challenges: The “Curse of Dimensionality”

High-dimensional data confronts cluster analysis with several problems. A bundle of problems is commonly addressed as the “curse of dimensionality”. Aspects of this “curse” most relevant to the clustering problem are: (i) In general, any optimization problem becomes increasingly difficult with an increasing number of variables (attributes) [5]. (ii) The relative contrast of the farthest point and the nearest point converges to 0 with increasing data dimensionality [6,9], i.e., the discrimination between the nearest and the farthest neighbor becomes rather poor in high dimensional data spaces. (iii) Capabilities of automated data acquisition in many application domains leads to the collection of as many features as possible in the expectation that many of these features may sometimes provide useful insights. So, for the task at hand, in many problems there exist many irrelevant attributes in a data set. Since groups of data are defined by some of the attributes only, the remaining irrelevant attributes (“noise”) may heavily interfere with the efforts to find these groups. (iv) Similarly, in a data set containing many attributes, some attributes will most probably exhibit correlations among each other (in varying complexity).

Many approaches try to alleviate the “curse of dimensionality” by applying feature reduction methods prior to cluster analysis. However, the second main challenge for cluster analysis of high dimensional data is the possibility, and even high probability, that different subsets or combinations of attributes may be relevant for different clusters. Thus, a global feature selection or dimensionality reduction method cannot be applied. Rather, it becomes an intrinsic problem of the clustering approach to find the relevant subspaces and to find clusters in these relevant subspaces. Furthermore, although correlation among attributes is often the basis for a dimension reduction, for many application domains it is a main part of the interesting information what correlations exist among which attributes for which subset of objects. As a consequence of this second challenge, the first challenge (i.e., the “curse of dimensionality”) generally cannot be alleviated for clustering high dimensional data.

Different Solutions: Categories of Subspace Clustering Techniques

Subspace clustering techniques can be divided into three main families. In view of the challenges sketched above, any arbitrarily oriented subspace may be interesting for a subspace clustering approach. The most general techniques (“(arbitrarily) oriented clustering”, “correlation clustering” (Note that the name “correlation clustering” relates to a different problem within the machine learning community.)) tackle this infinite search space. Yet most of the research in this field assumes the search space to be restricted to axis-parallel subspaces. Since the search space of all possible axis-parallel subspaces of a d -dimensional data space is still in $O(2^d)$, different search strategies and heuristics are implemented. Axis-parallel approaches mainly split into “subspace clustering” and “projected clustering”. In between these two main fields a group of approaches is known as “pattern-based clustering” (also: “biclustering” or “co-clustering”). For these approaches, the search space is not necessarily restricted to axis-parallel subspaces, but on the other hand does not contain all arbitrarily oriented subspaces. The restrictions on the search space differ substantially between different approaches in this group.

Axis-Parallel Subspaces

To navigate through the search space of all possible axis-parallel subspaces and to find clusters in subspaces, mainly two strategies are implemented: the top-down approach and the bottom-up approach.

Following the top-down approach, an algorithm derives a cluster approximately based on the full-dimensional space, and refines the cluster by adapting the corresponding subspace based on the current selection of points. This means, a lower dimensional projection is sought for where the (iteratively refined) set of points clusters best. Thus, algorithms pursuing this approach are called “projected clustering algorithms” and, usually, assign each point to at most one subspace cluster. The first approach of this category is proposed in [2].

Bottom-up approaches start by single dimensions and search primarily for all interesting subspaces (i.e., subspaces containing clusters) as combinations of lower dimensional interesting subspaces (often this combination is translated to the frequent item set problem and, thus, based on the A priori property). Most of these approaches are therefore “subspace clustering algorithms”, and can usually assign one point to

different clusters simultaneously (i.e., subspace clusters may overlap). Their aim is to find all clusters in all subspaces. There are also “hybrid algorithms” following the projected clustering approach but allowing points to belong to multiple clusters simultaneously or, on the other hand, following the subspace clustering approach but not computing all clusters in all subspaces. The first approach in this category is proposed in [4].

In summary, approaches to axis-parallel subspace clustering handle the problem of irrelevant attributes (aspect (iii) of the “curse of dimensionality”). Bottom-up-approaches, additionally, tackle mostly the problem of poor discrimination of nearest and farthest neighbor (aspect (ii)).

Pattern-Based Clustering

Pattern-based clustering algorithms seek subsets of objects exhibiting a certain pattern on a subset of attributes. In the most-spread algorithms, this pattern is an additive model of the cluster, meaning, each attribute value within a cluster and within the relevant subset of attributes is given by the sum of a cluster mean value, and an adjustment value for the current object and an adjustment value for the current attribute. In general, covering a cluster with such an additive model is possible if the contributing attributes exhibit a simple linear positive correlation among each other. This excludes negative or complex correlations, thus restricting the general search space. Cluster objects reside sparsely on hyperplanes parallel to the irrelevant axes. Projected onto the relevant subspace, the clusters appear as increasing one-dimensional lines. In comparison to axis-parallel approaches, the generalization consists mainly in allowing the axis-parallel hyperplane to be sparse. Also the cluster in the projection subspace may remain sparse. The unifying property of all cluster members is the common pattern. This model has basically been introduced in [7].

Allowing sparseness in the spatial patterns is an interesting feature of this family of approaches, since this also alleviates aspects (ii) and (iii) of the “curse of dimensionality”. Aspect (iv) is addressed partially.

Correlation Clustering

Correlation clustering approaches follow the most general model: Points forming a cluster can be located on an arbitrarily oriented hyperplane (i.e., subspace). These patterns occur if some attributes follow linear,

but complex correlations among each other (i.e., one attribute may be the linear combination of several other attributes). The main point addressed by these approaches is therefore aspect (iv) of the “curse of dimensionality”. The most widespread technique is the application of principal component analysis (PCA) on locally selected sets of points. Other techniques are the concept of the fractal dimension or applying the Hough transform on the data set. The first approach is proposed in [3]. The general model for this family of approaches is described in [1].

Key Applications

In many scientific and economic fields (like astronomy, physics, medicine, biology, archaeology, geology, geography, psychology, and marketing) vast amounts of high dimensional data are collected. To gain the full potential out of the gathered information, subspace clustering techniques are useful in all these domains. Pattern-based approaches are especially popular in microarray data analysis.

Future Directions

The different groups of subspace clustering techniques (subspace clustering, projected clustering, pattern-based clustering, correlation clustering) tackle different subproblems of the “curse of dimensionality”. There remain challenges for each of these problems. However, as a next-generation-type of approach, algorithms to tackle more and more aspects simultaneously can be expected.

Cross-references

- ▶ [Apriori Property and Breadth-First Search Algorithms](#)
- ▶ [Clustering Overview and Applications](#)
- ▶ [Curse of Dimensionality](#)
- ▶ [Data Mining](#)
- ▶ [Dimensionality Reduction](#)
- ▶ [Feature Selection for Clustering](#)

Recommended Reading

1. Achtert E., Böhm C., Kriegel H.-P., Kröger P., and Zimek A. Deriving quantitative models for correlation clusters. In Proc. 12th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2006.
2. Aggarwal C.C., Procopiuc C.M., Wolf J.L., Yu P.S., and Park J.S. Fast algorithms for projected clustering. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
3. Aggarwal C.C., and Yu P.S. Finding generalized projected clusters in high dimensional space. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000.

4. Agrawal R., Gehrke J., Gunopulos D., and Raghavan P. Automatic subspace clustering of high dimensional data for data mining applications. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998.
5. Bellman R. Adaptive Control Processes. A Guided Tour. Princeton University Press, 1961.
6. Beyer K., Goldstein J., Ramakrishnan R., and Shaft U. When is “nearest neighbor” meaningful? In Proc. 7th Int. Conf. on Database Theory, 1999.
7. Cheng Y. and Chruch G.M. Bioclustering of expression data. In Proc. 8th Int. Conf. Intelligent Systems for Molecular Biology, 2000.
8. Hartigan J.A. Direct clustering of a data matrix. *J. Am. Stat. Assoc.*, 67(337):123–129, 1972.
9. Hinneburg A., Aggrawal C.C., and Keim D.A. What is the nearest neighbor in high dimensional spaces? In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
10. Kriegel H.P., Kröger P., and Zimek A. Clustering high dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1), 2009.
11. Madeira S.C. and Oliveira A.L. Bioclustering algorithms for biological data analysis: A survey, *IEEE Trans. Comput. Biol. Bioinf.*, 1(1):24–45, 2004.
12. Parsons L., Haque E., and Liu H. Subspace clustering for high dimensional data: A review. *SIGKDD Explorations*, 6(1):90–105, 2004.

Subspace Selection

- ▶ [Dimension Reduction Techniques for Clustering](#)

Subsumed by Windows Communication Framework

- ▶ [.NET Remoting](#)

Success at n

NICK CRASWELL
Microsoft Research, Cambridge, UK

Synonyms

S@n

Definition

Success at n is an information retrieval relevance measure, equal to 1 if the top-n documents contain a

relevant document and 0 otherwise. When averaged across multiple queries, the success rate at n indicates how often something relevant was retrieved within the top- n .

Key Points

A system with a high success rate will be one that rarely retrieves zero relevant documents. Therefore, success rate can be employed to monitor failure. The success rate of two systems may differ even if they have the same mean precision at n , because one system has higher variance than the other. Success at n models the satisfaction of a user who does not need to see many relevant documents, is prepared to view up to n results and is disappointed by a completely irrelevant top- n .

Cross-references

- ▶ Precision at n
- ▶ Precision-Oriented Effectiveness Measures

Succinct Constraints

CARSON KAI-SANG LEUNG

University of Manitoba, Winnipeg, MB, Canada

Definition

Let Item be the set of domain items. Then, an itemset $SS_j \subseteq \text{Item}$ is a *succinct set* if SS_j can be expressed as a result of selection operation $\sigma_p(\text{Item})$, where σ is the usual selection operator and p is a selection predicate. A powerset of items $SP \subseteq 2^{\text{Item}}$ is a *succinct powerset* if there is a fixed number of succinct sets $SS_1, \dots, SS_k \subseteq \text{Item}$ such that SP can be expressed in terms of the powersets of SS_1, \dots, SS_k using set union and/or set difference operators. A constraint C is *succinct* provided that the set of itemsets satisfying C is a succinct powerset.

Key Points

Succinct constraints [1,2] possess the following nice properties. For any succinct constraint C , there exists a precise “formula” – called a *member generating function (MGF)* – to enumerate *all* and *only* those itemsets that are guaranteed to satisfy C . Hence, if C is succinct, then C is pre-counting prunable. This means that one can directly generate precisely the itemsets that satisfy

C – without looking at the transaction database TDB and even before counting the support (or frequency) of itemsets. In other words, whether an itemset S satisfies C or not can be determined based on the selection of items from the domain. Examples of succinct constraints include $\max(S.\text{Price}) \leq \120 and $\max(S.\text{Price}) \geq \80 , which express that the maximum price of all items in an itemset S is at most \$120 and at least \$80 respectively. The set of itemsets satisfying the former can be expressed as $2^{\sigma_{\text{Price} \leq \$120}(\text{Item})}$; these itemsets can be enumerated by using only the items having prices at most \$120 – via the MGF $\{X \mid X \subseteq \sigma_{\text{Price} \leq \$120}(\text{Item}), X \neq \emptyset\}$. Similarly, the set of itemsets satisfying the second constraint $\max(S.\text{Price}) \geq \80 can be expressed as $2^{\text{Item}} - 2^{\sigma_{\text{Price} < \$80}(\text{Item})}$; these itemsets can be enumerated by using the items having prices at least \$80 (i.e., mandatory items) and other items (i.e., optional items) – via the MGF $\{Y \cup Z \mid Y \subseteq \sigma_{\text{Price} \geq \$80}(\text{Item}), Y \neq \emptyset, Z \subseteq \sigma_{\text{Price} < \$80}(\text{Item})\}$.

Cross-references

- ▶ Frequent Itemset Mining with Constraints

Recommended Reading

1. Lakshmanan L.V.S., Leung C.K.-S., and Ng R.T. Efficient dynamic mining of constrained frequent sets. *ACM Trans. Database Syst.*, 28(4):337–389, 2003.
2. Ng R.T., Lakshmanan L.V.S., Han J., and Pang A. Exploratory mining and pruning optimizations of constrained associations rules. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 13–24.

Suffix Stripping

- ▶ Stemming

Suffix Tree

MAXIME CROCHEMORE^{1,2}, THIERRY LECROQ³

¹King's College London, London, UK

²University of Paris-East, Paris, France

³University of Rouen, Rouen, France

Synonyms

Compact suffix tries

Definition

The suffix tree $\mathcal{S}(y)$ of a non-empty string y of length n is a compact trie representing all the suffixes of the string.

The suffix tree of y is defined by the following properties:

1. All branches of $\mathcal{S}(y)$ are labeled by all nonempty suffixes of $y \sqcup$.
2. Edges of $\mathcal{S}(y)$ are labeled by strings.
3. Internal nodes of $\mathcal{S}(y)$ have at least two children.
4. Edges outgoing an internal node are labeled by segments starting with different letters.
5. The segments are represented both by their starting position on y and their lengths.

The space sign \sqcup appended at the end of y avoids marking nodes, and implies that $\mathcal{S}(y)$ has exactly n leaves (number of non-empty suffixes).

All the properties then imply that the total size of $\mathcal{S}(y)$ is $O(n)$, which makes it possible to design a linear-time construction of the suffix tree.

Historical Background

The first linear time algorithm for building a suffix tree of a string of length n is from Weiner [15], but it requires quadratic space: $O(n \times \sigma)$ where σ is the size of the alphabet. The first linear time and space algorithm for building a suffix tree is from McCreight [13]. It works “off-line,” inserting the suffixes from the longest one to the shortest one. A strictly sequential version of the suffix tree construction was described by Ukkonen [14]. When the alphabet is potentially infinite, the optimal construction algorithms of the suffix tree can be implemented to run in time $O(n \log \sigma)$, since they imply an ordering on the letters of the alphabet. On particular integer alphabets, Farach [5] showed that the construction can be done in linear time.

The minimization in the sense of automata theory of the suffix trie gives the suffix automaton. The suffix automaton of a string is also known under the name of DAWG, for *Directed Acyclic Word Graph*. Its linearity was discovered by Blumer et al. (see [1]), who gave a linear construction (on a fixed alphabet). The minimality of the structure as an automaton is from Crochemore [2], who showed how to build the factor automaton of a text with the same complexity.

The compaction of the suffix automaton gives the compact suffix automaton (see [1]). The compaction consists of removing all internal nodes with only one

child and concatenating remaining successive edge labels. A direct construction algorithm of the compact suffix automaton was presented by Crochemore and Vérin [4]. The same structure arises when minimizing the suffix tree.

The suffix array of the string y consists of both the permutation of positions on the text that gives the sorted list of suffixes, and the corresponding array of lengths of their longest common prefixes (LCP). The suffix array of a string with the associated search algorithm based on the knowledge of the common prefixes is from Manber and Myers [12]. The suffix array can be built in linear time on integer alphabets (see [8–10]).

For the implementation of index structures in external memory, the reader can refer to Ferragina and Grossi [6].

Foundations

Suffix Trees

The suffix tree $\mathcal{S}(y)$ of the string $y = ababbb\sqcup$ is presented in Fig. 1. It can be seen as a compaction of the suffix trie $\mathcal{T}(y)$ of y given in Fig. 2.

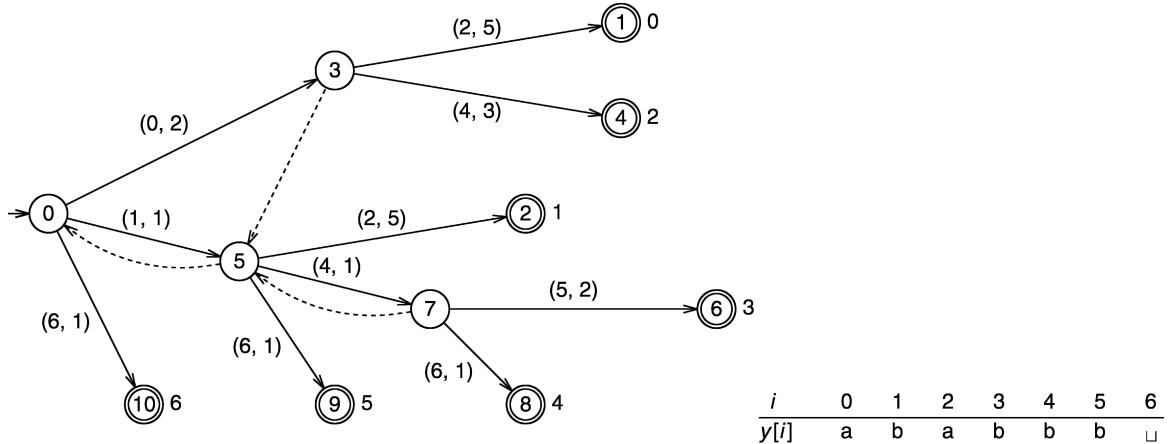
Nodes of $\mathcal{S}(y)$ and $\mathcal{T}(y)$ are identified with segments of y . Leaves of $\mathcal{S}(y)$ and $\mathcal{T}(y)$ are identified with suffixes of y . An output is defined, for each leaf, which is the starting position of the suffix in y .

The two structures can be built by successively inserting the suffixes of y from the longest to the shortest.

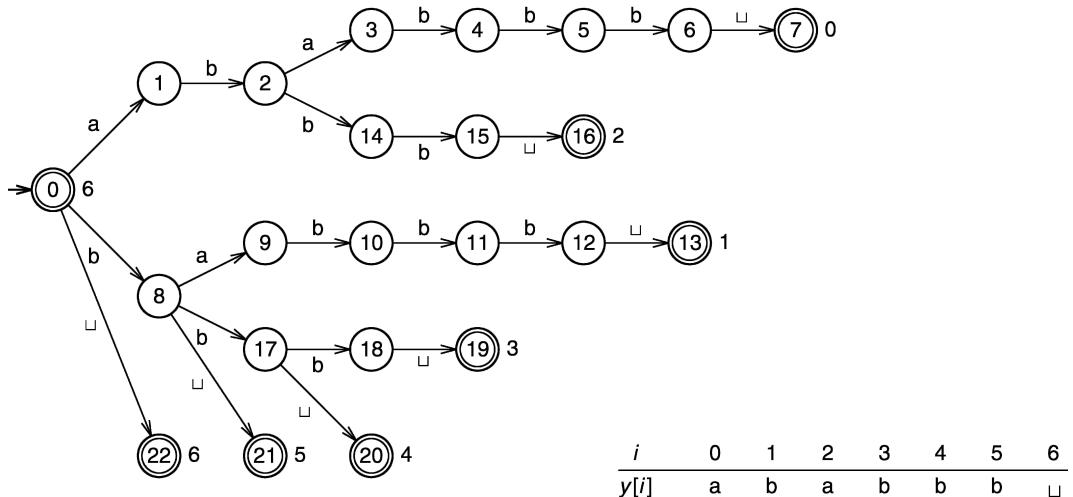
In the suffix trie $\mathcal{T}(y)$ of a string y of length n there exist n paths from the root to the n leaves: each path spells a different non-empty suffix of y . Edges are labeled by exactly one symbol. The suffix trie can have a quadratic number of nodes, since the sum of the lengths of all the suffixes of y is quadratic.

To get the suffix tree from the suffix trie, internal nodes with exactly one successor are removed. Labels of edges between remaining nodes are then concatenated. Edges are now labeled with strings. This gives a linear number of nodes, since there are exactly n leaves and since every internal node (called a fork) has at least two successors; there can be at most $n - 1$ forks. This also gives a linear number of edges.

Now, in order that the space requirement becomes linear, since the labels of the edges are all segments of y , a segment $y[i .. i + \ell - 1]$ is represented by the pair



Suffix Tree. Figure 1. Suffix tree $S(y)$ of $ababbb$. Nodes are numbered in the order of creation. The small number close to each leaf corresponds to the position of the suffix associated with the leaf.



Suffix Tree. Figure 2. Suffix trie $T(y)$ of $ababbb$. Nodes are numbered in the order of creation. The small number close to each leaf corresponds to the position of the suffix associated with the leaf.

(i, ℓ) . Thus, each edge can be represented in constant space. This technique requires having y residing in the main memory.

Overall, there is a linear-number of nodes and a linear number of edges, each node and each edge can be represented in constant space, thus, the suffix tree requires linear space.

There exist several direct linear-time-construction algorithms of the suffix tree that avoid the construction of the suffix trie followed by its compaction.

The McCreight's algorithm [13] directly constructs the suffix tree of the string y by successively inserting

the suffixes of y from the longest one to the shortest one. The insertion of the suffix of y beginning at position i (i.e., $y[i..n - 1]$) consists first of locating (creating it if necessary) the fork associated with the longest prefix of $y[i..n - 1]$ common with a longer suffix of y : $y[0..n - 1], y[1..n - 1], \dots$, or $y[i - 1..n - 1]$. Let us call the head u this longest prefix and call the tail v the remaining part of the suffix such that $y[i..n - 1] = uv$. Once the fork p associated with u has been located, it is enough to add a new leaf q labeled by i and a new edge labeled by $(i + |u|, |v|)$ from p to q to complete the insertion of $y[i..n - 1]$ into the

structure. The reader can refer to [3] or [7] for further details.

The linear time of the construction is achieved by using a function called suffix link, defined on the forks as follows: if fork p is identified with segment av , $a \in A$ and $v \in V^*$, then $s_y(p) = q$ where fork q is identified with v .

Suffix links are represented by dotted arrows in Fig. 1.

The suffix links create shortcuts that are used to accelerate head computations. If the head of $y[i - 1 .. n - 1]$ is of the form au ($a \in A$, $u \in V^*$) then u is a prefix of the head of $y[i .. n - 1]$. Therefore, using suffix links, the insertion of the suffix $y[i .. n - 1]$ consists first of finding the fork corresponding to the head of $y[i .. n - 1]$ (starting from suffix link of the fork associated with au), and then in inserting the tail of $y[i .. n - 1]$ from this fork.

Ukkonen algorithm [14] works on-line, i.e., it builds the suffix tree of y processing the symbols of y from the first to the last. It also uses suffix links to achieve a linear-time computation.

Weiner, McCreight and Ukkonen algorithms work in $O(n)$ time whenever $O(n \times \sigma)$ space is used. If only $O(n)$ space is used then the $O(n)$ time bound should be replaced by $O(n \times \log \min\{n, \sigma\})$. This accounts for the time to access a specific edge stored in each nodes. The reader can refer to [11] for specific optimized implementations of suffix trees.

For particular integer alphabets, when the alphabet of y is in the interval $[1 .. n^c]$ for some constant c , Farach [5] showed that the construction can be done in linear time.

Generalized suffix trees are used to represent all the suffixes of all the strings belonging to a set of strings.

Indexes

The suffix tree serves as a full index on the string: it provides a direct access to all segments of the string, and gives the positions of all their occurrences in the string. An index on y can be considered as an abstract data type whose basic set is the set of all the segments of y , and that possesses operations giving access to information relative to these segments. The utility of considering the suffixes of a string for this kind of application comes from the obvious remark that every segment of a string is the prefix of a suffix of the string.

Once the suffix tree of a text y is built, searching for x in y remains to spell x along a branch of the tree. If this walk is successful, the positions of the pattern can be output. Otherwise, x does not occur in y .

Any kind of trie that represents the suffixes of a string can be used to search it. However, the suffix tree has additional features which imply that its size is linear.

We consider four operations relative to the segments of a string y : membership, first position, number of occurrences and list of positions.

The first operation on an index is the membership of a string x to the index, that is to say the question to know whether x is a segment of y . This question can be specified in two complementary ways according to whether x is a segment of y or not. If x does not occur in y , it is often interesting in practice to know the longest beginning of x that is a segment of y . This is the type of usual answer necessary for the sequential search tools in a text editor.

The methods produce without large modification the position of an occurrence of x , and even the position of the first or last occurrence of x in y .

Knowing that x is in the index, other relevant information is constituted by its number of occurrences in y . This information can differently direct the ulterior searches.

Finally, with the same assumption as previously, complete information on the localization of x in y is supplied by the list of positions of its occurrences.

Suffix trees can easily answer these questions. It is enough to spell x from the root of $\mathcal{S}(y)$. If it is not possible, then x does not occur in y . Whenever x occurs in y , let w be the shortest segment of y that is such x is a prefix of w and w is associated with a node p of $\mathcal{S}(y)$. The number of leaves of the subtree rooted in the node p then gives the number of occurrences of x in y . All these numbers associated with nodes of the tree can be precomputed in linear time. The smallest (respectively largest) of these leaves gives the position of the first (resp. last) position of x in y . The list of leaves numbers gives the list of the position of x in y (see [3]).

Key Applications

Suffix trees are used to solve string searching problems, mainly when the text into which several patterns have to be found is fixed. It is also used in other string related problems such as Longest Repeated

Substring, Longest Common Substring. It can be used to perform text compression. Word suffix trees can be used when processing natural languages, in order to represent only suffixes starting after separators such as space or line feed. Gusfield [7] gives many applications of suffix trees in computational biology.

Cross-references

► Tries

Recommended Reading

1. Blumer A., Blumer J., Ehrenfeucht A., Haussler D., Chen M.T., and Seiferas J. The smallest automaton recognizing the subwords of a text. *Theor. Comput. Sci.* 40(1):31–55, 1985.
2. Crochemore M. Transducers and repetitions. *Theor. Comput. Sci.* 45(1):63–86, 1986.
3. Crochemore M., Hancart C., and Lecroq T. *Algorithms on Strings*. Cambridge University Press, Cambridge, UK, 2007.
4. Crochemore M. and Vérit R. On compact directed acyclic word graphs. In *Structures in Logic et Computer Science*, LNCS 1261: 192–211, 1997.
5. Farach M. Optimal suffix tree construction with large alphabets. In Proc. 38th IEEE Annual Symp. on Foundations of Computer Science, 1997, pp. 137–143.
6. Ferragina P. and Grossi R. The string B-tree: A new data structure for string search in external memory et its applications. *J. Assoc. Comput. Mach.*, 46:236–280, 1999.
7. Gusfield D. *Algorithms on strings, trees and sequences*. Cambridge University Press, Cambridge, UK, 1997.
8. Kärkkäinen J. and Sanders P. Simple linear work suffix array construction. In Proc. 30th Int. Colloquium on Automata, Languages, and Programming, 2003, pp. 943–955.
9. Kim D.K., Sim J.S., Park H., and Park K. Linear-time construction of suffix arrays. In Proc. 14th Annual Symp. Combinatorial Pattern Matching, 2003, pp. 186–199.
10. Ko P. and Aluru S. Space efficient linear time construction of suffix arrays. In Proc. 14th Annual Symp. Combinatorial Pattern Matching, 2003, pp. 200–210.
11. Kurtz S. Reducing the space requirement of suffix trees. *Softw., Pract. Exp.* 29(13):1149–1171, 1999.
12. Manber U. and Myers G. Suffix arrays: a new method for on-line string searches. *SIAM J. Comput.* 22(5):935–948, 1993.
13. McCreight E.M. A space-economical suffix tree construction algorithm. *J. Algorithms* 23(2):262–272, 1976.
14. Ukkonen E. On-line construction of suffix trees. *Algorithmica* 14(3):249–260, 1995.
15. Weiner P. Linear pattern matching algorithm. In Proc. 14th Annual IEEE Symp. on Switching et Automata Theory. Washington, DC, 1973, pp. 1–11.

Suffixing

► Stemming

Summarizability

ARIE SHOSHANI

Lawrence Berkeley National Laboratory, Berkeley, CA, USA

Synonyms

[Summarization correctness](#); [Statistical correctness](#)

Definition

Summarizability is a property that assures the correctness of summary operations over *On-Line Analytical Processing* (OLAP) databases, which are akin to Statistical Databases [10]. Such databases are generally referred to as “summary databases,” and have a data model based on one or more measures defined over the cross product of *dimensions*. For example, a bookstore company may have multiple stores in many cities. Assume that there is a database containing the stores revenues for books sold per day over the last three years. In such a database, “revenue” is a *measure*, and “book,” “store,” “day” are the dimensions that define the cross product over which the measure revenue is defined. A dimension in a summary database is said to be *summarizable* relative to a measure, if a summary statistic (sum, average, etc.) applied over the dimension produces correct results. For example, if summarization over all the books sold to obtain “total_revenues per store, per day” yields correct results, the dimension “book” is considered summarizable relative to the measure “revenue.” There are certain conditions that have to hold in order to get correct results, which are discussed in the next section. Often, dimensions are organized into a *hierarchy* of categories. For example, days can naturally be organized into months, and months into years. Similarly, books can be organized by book-types (e.g., cooking, fiction, etc.). Summarization can then be applied to categories, such as summarizing over books and days to get “revenue per book-type, per store, per year.” In such cases the summarizability property must apply to each category level of the category hierarchy of a dimension, for that dimension to be considered summarizable.

Historical Background

Statistical Databases, which were introduced in the 1980s [2], and OLAP databases, introduced in the 1990s [1,3,4]

have a similar data model [1], but the issue of summarizability was not introduced until 1990 [9] and studied carefully until 1997 [7]. After that time, several authors have treated summarizability formally [5,6,8].

Foundations

Next, examples that violate summarizability are presented, and using these examples the conditions for summarizability are stated. There are three such conditions that must hold in order for a dimension to be summarizable. Two of the conditions refer to the category levels in a dimension, and the third is a condition of the dimension relative to the measure. Next, the basic notation used throughout this document is introduced.

Notation

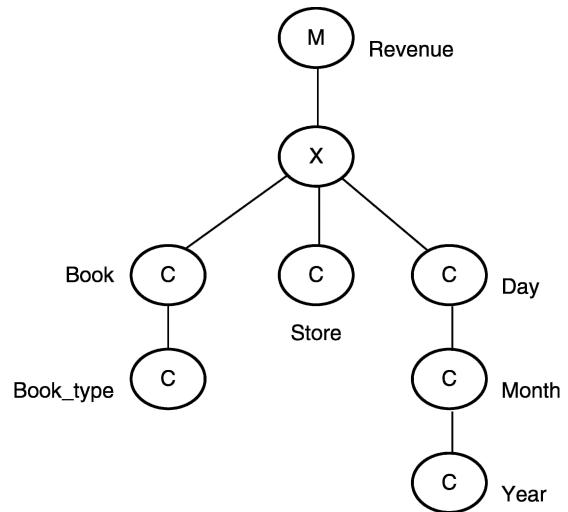
Consider, again, the example above of revenues per book, store, and day. Using a notation commonly used for such databases, this example database can be represented as “revenue (book, store, day).” For the category hierarchies of dimensions, the notation $[C_1 \rightarrow C_2 \rightarrow \dots C_i \rightarrow C_{i+1}, \dots C_n]$ is used to represent a category hierarchy of a dimension of height n , starting from the more detailed level towards higher levels. Thus, for the example above, where the two hierarchies mentioned above are over the dimensions book and day, the database will be represented as:

“revenue ([book-> book-type], store, [day -> month -> year]).”

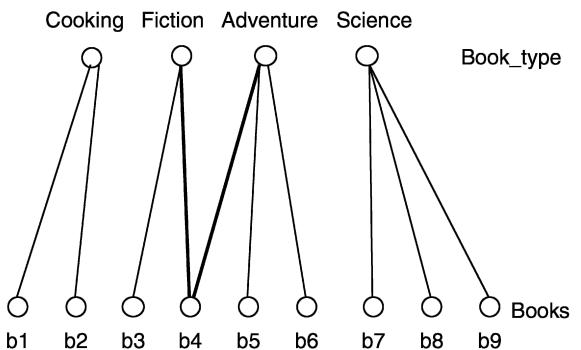
These concepts and notation are shown graphically in Fig. 1, where the letters M, C, and X represent Measure, Category-level, and X-product (cross-product), respectively.

The “Disjointness” Condition

Consider the revenue database above. Suppose that the book-type set is (cooking, fiction, adventure, science, etc.). Most of the books will usually belong to a single book_type, but some could be categorized under two or more types. This is shown graphically in Fig. 2, for sales in a particular day for a particular store. As can be seen, there in one book, b4, which is classified under the categories “fiction” and “adventure.” If the revenues by book_type are added to generate “revenue (book-type, store, day),” the totals will be incorrect, because if the revenues for all book-types are added, the revenue for book b4 is added twice. The reason is, of course, that book b4 belongs to two parent categories.



Summarizability. Figure 1. Revenue of books sold in a particular store on a particular day.



Summarizability. Figure 2. Books organized by book_type.

The disjointness condition states: given two consecutive category-levels C_{low} and C_{high} in a dimension, where $[C_{\text{low}} \rightarrow C_{\text{high}}]$, the sets of lower-level categories elements that belong to each category element of the higher-level, must be disjoint. This condition can also be expressed as requiring that the category elements of the category levels form a “strict” hierarchy [8]. Yet another way to state this condition is to say that there must be a one-to-many relationship from C_{high} to C_{low} .

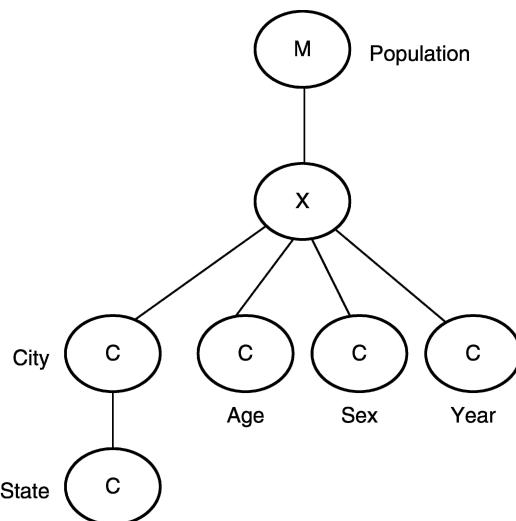
This seemingly simple observation is the source of incorrect statistics in many systems that do not enforce summarizability conditions. Under such conditions, summarization is still possible by special treatment, such as choosing to assign revenues equally to shared

nodes (in the example above assigning half the revenue for book b4 to each of the “fiction” and “adventure” book types). However, this is not usually done. Note that summarization of book revenues to get, for example, “revenues (store, day)” will yield a correct result, since the category book_type is not involved.

The “Completeness” Condition

Completeness is a condition that holds if all the children of higher-level category elements exist. If some of the children are missing, then the summary to the higher level may be incorrect. Consider, for example, a database that contains “population (city, year, race, sex).” Suppose further that cities are organized by states, as shown in Fig. 3. In this database, if the population is summarized to the “state” level, the result of populations is obviously incorrect, since only populations of cities are taken into account and not populations of villages and small towns. However, if the measure was stated as “populations_in_cities” then the [city -> state] category mapping would be summarizable. This example shows that the second condition of completeness is relative to the measure semantics.

One way to overcome the “incompleteness” condition is to add instances that account for the missing elements in the category. In the population example, one can add for each state, in addition to the cities in the state, an instance that accounts for the population in all areas other than cities. Such a node can be labeled “other_areas” for example. If this is done,



Summarizability. Figure 3. The population database is not summarizable to the “state” level.

summarization to the state level would yield the correct summary population.

Another way to determine if a category level satisfies the completeness condition can often be based on external knowledge. For example, suppose that the dimension “age” in Fig. 3 is organized into age_groups: (0–10) (10–20), ..., (90–100). Is summarization to the age_group level correct in this case? It is only correct if there is external knowledge that this database does not contain people older than 100. Otherwise, a category (>100) has to be added in order to satisfy the completeness condition.

The “Measure Type” Condition

Consider the database in Fig. 3 again, where a higher category level is added to the dimension year: [year -> decade]. Obviously, population cannot be summarized to the decade level, since adding the yearly populations does not yield a meaningful measure for the decade. However, if the measure was “average population” per year (and “counts” were also recorded), the average population per decade could be calculated. Why is that? As another example, consider the book revenues database in Fig. 1. Obviously, the revenues can be summarized from days, to months, to years. However, if the measure was “number_of_unsold_books,” this cannot be summarized (added) over the time dimension. The reason stems from the semantic behavior of “temporal aggregation.”

In statistics the term “temporal aggregation” is used to describe the behavior of measures when aggregating over the time domain. The measures are classified into three types: “stock,” “flow,” and “value-per-unit.” It turns out that these types behave differently when summarized over time, depending on the summary statistics used. In particular, a measure of “stock” type cannot be summed over the time domain, whereas a measure of “flow” type can be summed over the time domain. In the example discussed above, “population” is of type stock, and so is “number_of_unsold_books,” and therefore they cannot be summed over the time dimension. In contrast, “book_revenues” is of type “flow,” and therefore can be summed.

In general, measures of type “stock” refer to a state of the measure recorded at a particular point in time (such as inventory), while measures of type “flow” record values of events over a period of time (such as sales). A measure of the type “value-per-unit” is similar to “stock” in that it is recorded at a particular point in time, but it has a per-unit value (such as the

cost of a book). In [7] a table is given for temporal summarizability for each measure type for five common aggregation operators: min, max, sum, avg, and range. The table is reproduced in Fig. 4. As can be seen only the operator “sum” is not summarizable for the types “stock” and “value-per-unit.” It turns out that “value-per-unit” is also not summarizable for non-temporal aggregation in the case of “sum,” but all other cases are summarizable for non-temporal aggregation [7].

Summary

The conditions that are necessary to ensure correctness of aggregation operations over statistical and OLAP databases were presented. Such conditions are referred to as “summarizability conditions.” Summarizability conditions apply to the dimensions of multidimensional data structures and to the category hierarchies in each dimension. Such conditions depend on the summary measure type, and whether the summarization is over the time domain. Note that the summarizability of each category level in a hierarchy is independent of the others; that is, some category levels can be summarizable while others are not.

Summarizability conditions are applicable to any database system that supports aggregation operations. While these conditions were described here in the context of the OLAP data model, these conditions apply to other models that do not express the multidimensional structures explicitly. In particular, it is possible to represent an OLAP schema as a relational schema, where the semantics of multidimensionality and category hierarchies are not explicit. In order to achieve correct results in aggregation operations from such relational databases, it is necessary to identify these (multidimensionality and category hierarchies) structures, and to make sure that the summarizability conditions hold. If all the conditions do not hold, aggregation operations should be avoided and/or refused.

	stock	flow	value/unit
min	yes	yes	yes
max	yes	yes	yes
sum	no	yes	no
avg	yes	yes	yes
range	yes	yes	yes

Summarizability. Figure 4. Temporal summarizability by measure type and function type.

Key Applications

It is often claimed that statistical operations can be inaccurate for various reasons. The better-known reason is summarization over null values. For example, taking an average over a set of values where some are null, will produce the wrong result if the null elements are represented as zeros, or if they are not discounted in the computation. Summarizability conditions are just as important, but are more subtle, semantically based, and are often overlooked. It is essential that such conditions are checked in any database that provides aggregation operators, including OLAP and relational database systems.

Future Directions

The three conditions described above for ensuring summarizability are necessary conditions. However, while it is believed that these conditions are also sufficient, this was not shown formally so far. Another aspect of future work is adding annotations to schemas as to whether summarizability conditions hold, and how to automate the checking of summarizability conditions dynamically in a database as data instances are entered into (or modified in) the database. Once summarizability is made part of the data model, it is necessary to enhance the aggregation operators to avoid summarization over non-summarizable data.

Cross-references

- Dimension
- Hierarchy
- Measure
- Multidimensional Modeling
- On-Line Analytical Processing
- Quality of Data Warehouses
- Query Processing in Data Warehouses
- Scientific Databases

Recommended Reading

1. Agrawal R., Gupta A., and Sarawagi S. Modeling multidimensional databases. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 232–243.
2. Chan P. and Shoshani A. Subject: a directory driven system for organizing and accessing large statistical databases. In Proc. 7th Int. Conf. on Very Data Bases, 1981, pp. 553–563.
3. Codd E.F., Codd S.B., and Salley C.T. Providing olap (online analytical processing) to user-analysts: An IT mandate, Codd and Associates technical report, 1993.
4. Gray J., Bosworth A., Layman A., and Pirahesh H. Data cube: a relational aggregation operator generalizing group-by, cross-tabs

- and sub-totals. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 152–159.
5. Hurtado C.A., Gutiérrez C., and Mendelzon A. Capturing summarizability with integrity constraints in OLAP. *ACM Trans. Database Syst.*, 30(3):854–886, 2005.
 6. Hurtado C.A. and Mendelzon A.O. Reasoning about summarizability in heterogeneous multidimensional schemas. In Proc. 8th Int. Conf. on Database Theory, 2001, pp. 375–389.
 7. Lenz H-J. and Shoshani A. Summarizability in OLAP and statistical data bases. In Proc. 9th Int. Conf. on Scientific and Statistical Database Management, 1997, pp. 132–143.
 8. Pedersen T.B. and Jensen C.S. Multidimensional data modeling for complex data. In Proc. 15th Int. Conf. on Data Engineering, 336–345, 1999.
 9. Rafanelli M. and Shoshani A. STORM: a statistical object representation model. In Proc. 2nd Int. Conf. on Scientific and Statistical Database Management, 1990, pp. 14–29.
 10. Shoshani A. OLAP and statistical databases: similarities and differences. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 185–196.

Summarization

JIMMY LIN

University of Maryland, College Park, MD, USA

Synonyms

[Text/document summarization](#); [Automatic abstracting](#); [Distillation](#); [Report writing](#)

Definition

Summarization systems generate condensed outputs that convey important information contained in one or more sources for particular users and tasks. In principle, input sources and system outputs are not limited to text (e.g., keyframe extraction for video summarization), but this entry focuses exclusively on generating *textual* summaries from *textual* sources.

Historical Background

Summarization has a long history dating back to the 1960s, when researchers first started developing computer systems that processed natural language [6,12]. Following a number of decades with comparatively few publications, summarization research entered a new phase in the 1990s. A revival of interest was spurred by the growing availability of text in electronic formats and later the World Wide Web. The enormous quantities of information people come into contact with on a daily basis created a need for applications that help

users cope with the proverbial information overload problem. Summarization systems attempt to address this need.

Foundations

Summarization is a broad and diverse field. Traditionally, it is considered a sub-area of natural language processing, but a significant number of innovations have their origins in information retrieval. This entry is organized as follows: first, various summarization factors are discussed. Next, a tripartite processing model for summarization systems is presented, which provides a basis for discussing general issues. Finally, selected summarization techniques are briefly overviewed.

Summarization Factors

To better understand summarization, it is helpful to enumerate its many dimensions – what Sparck Jones [19] calls “factors”. These factors provide a basis for understanding various automatic methods, and can be grouped into three broad categories: *input*, *purpose*, and *output*. What follows is meant to be an overview of important factors, and not intended to be exhaustive.

Input factors characterize the source of the summaries:

1. *Single versus multiple sources*. For example, one versus multiple reports of the same event.
2. *Genre* (categories of texts) and *Register* (different styles of writing). For example, dissertations versus blogs.
3. *Written versus spoken*. For example, newspaper articles versus broadcast news.
4. *Language*. Sources may be in multiple language.
5. *Metadata*. Sources may be associated with controlled vocabulary keywords, human-assigned category labels.
6. *Structure*. Source structure may be relatively straightforward (e.g., headings and sub-headings) or significantly more complex (e.g., email threads).

Purpose factors characterize the use of summaries (i.e., why they were created):

1. *Indicative versus informative versus evaluative*. Indicative summaries are meant to guide the selection of sources for more in-depth study, whereas informative summaries cover salient information in the sources at some level of detail (and is often meant to replace the original). Evaluative summaries

- assess the subject matter of the source and the quality of the work (e.g., a review of a movie).
2. *Generic versus focused.* A generic summary places equal emphasis on different information contained in the sources and provides balanced coverage. Alternatively, a summary might be focused on an information need, i.e., created to answer a question.
 3. *Task.* What will the summary be used for? For example, to help write a report or to make a decision.
 4. *Audience.* Whom is the summary intended for? For example, experts, schoolchildren, etc.

Output factors characterize system output (note that the input factors are relevant here also, but not repeated):

1. *Extractive versus abstractive.* Extractive summaries consist of text copied from the source material; typically, such approaches are based on shallow analysis. Abstractive summaries contain text that is system-generated, usually based on deeper analysis. Note that these approaches define a continuous spectrum, as many systems employ hybrid methods.
2. *Reduction, coverage, and fidelity.* Reduction, usually measured as a ratio between summary length and source length, is often inversely related to coverage, how much information of interest is preserved in the summary. The summary should also preserve source information accurately.
3. *Coherence.* Does the summary read fluently and grammatically, both syntactically and at the discourse level? For summaries not intended to be fluent prose (e.g., bullets), this factor is less important.

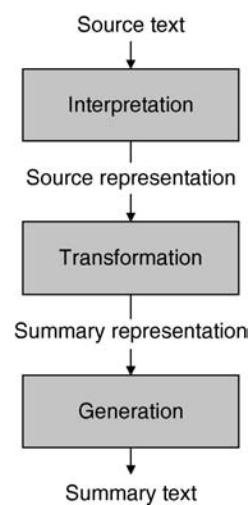
Input, purpose, and output factors together characterize the many dimensions of summarization and provide a basis for subsequent discussions. Note, however, that not all factors figure equally in current summarization systems – for a variety of reasons, the field has focused on some more than others.

Processing Model

Sparck Jones characterizes the process of summarization as a reductive transformation of source text to summary text through content condensation by selection and/or generalization of what is important in

the source [19]. She proposes a tripartite processing model, shown in Figure 1, that serves as a framework for understanding how various summarization techniques fit together (see also [15] for a similar model). Systems first convert source text into the source representation, which is then transformed into the summary representation. Finally, the summary representation is realized as natural language text. Note that these stages do not necessarily map to system components, as the processing model only describes abstract processing tasks. Since this model does not prescribe specific representations or particular processing methods, it is sufficiently general to describe a wide variety of summarization systems while at the same time highlighting important differences.

As previously discussed, input may come from one or multiple sources (the term “documents” is used generically, recognizing that sources may also be speech, email, etc.). Single-document summarization is challenging because simple baselines are often very difficult to improve upon. For example, since news articles are typically written in the “inverse pyramid” style (most important information first), the first sentence or paragraph makes an excellent summary. Frequently, longer documents (e.g., reports) contain “executive summaries”, which nicely capture important information in the documents. Multi-document summarization faces a different set of challenges, the most salient of which is the possibility of redundant information in the sources (e.g., multiple news articles



Summarization. Figure 1. A tripartite processing model for summarization.

about the same event). Frequently, the redundancy is not superficially obvious, but involves paraphrase (different syntactic structures, word choices, etc.). More complex are cases where the information partially overlaps or appears contradictory (e.g., different reports of death tolls). More generally, multi-document summarization requires systems to detect similarities and differences in text.

It is generally assumed that a summarization system is provided the source text. In cases where this is assumption is not met, information retrieval techniques may be used to first select the set of documents to summarize (from a larger collection of documents). However, since most systems assume that input sources are more or less relevant to the task at hand, they may not adequately cope with imperfect retrieval results.

The use of “representation” does not necessarily imply deep linguistic analysis or processing. In fact, most extractive summarization systems adopt a “bag of words” representation at both the source and summary end – that is, text is represented as a vector that has a feature for each word. This representation makes the obviously false assumption that word occurrences are independent and ignores the rich linguistic relationships present in text. Nevertheless, extractive techniques have proven to be effective in various summarization tasks.

With extractive techniques, generation is trivial since systems simply copy material from the source. However, pure extraction often leads to problems in overall coherence of the summary – a frequent issue concerns “dangling” anaphora. Sentences often contain pronouns, which lose their referents when extracted out of context. Worse yet, stitching together decontextualized extracts may lead to a misleading interpretation of anaphors (resulting in an inaccurate representation of source information, i.e., low fidelity). Similar issues exist with temporal expressions. Note that these problems become more severe in the multi-document case, since extracts are drawn from different sources. A general approach to addressing these issues involves post-processing extracts, for example, replacing pronouns with their antecedents, replacing relative temporal expression with actual dates, etc. Such techniques, however, can not be considered purely extractive (hence the observation that most systems are, in fact, hybrid).

In general, extractive systems can be characterized as “knowledge-poor”, which is contrasted against “knowledge-rich” approaches. While not synonymous,

abstractive methods tend to be associated with “knowledge-rich” approaches. They involve one or more of the following: detailed linguistic analysis on source text to produce richly annotated structures, incorporation of world knowledge to support the transformation process, or generation of fluent natural language text from abstract representations.

A canonical example of abstractive summarization involves integration with information extraction (IE) systems. Information extraction concerns the automatic identification and creation of template instances from natural language text based on some pre-defined structure. For example, a template for natural disasters might contain “slots” for type, damage, death toll, etc. An IE system would analyze text sources and automatically extract information to fill these templates, in effect, populating a structured database from free text. This process can be viewed as the interpretation stage in the summarization processing model, and the templates themselves can serve as the source representation. A summarization system can then combine information from multiple templates to generate a fluent summary (e.g., [18]).

Abstractive techniques face a number of major challenges, the biggest of which is the representation problem. Systems’ capabilities are constrained by the richness of their representations and their ability to generate such structures – systems cannot summarize what their representations cannot capture. In limited domains, it may be feasible to devise appropriate structures, but a general-purpose solution depends on open-domain semantic analysis. Systems that can truly “understand” natural language are beyond the capabilities of today’s technology.

Finally, coherence of system-generated text is one important output factor in summarization. Coherence is usually taken to mean fluent, grammatically correct prose that “reads well”. This is a tall order, mainly because coherence is very difficult to operationalize. While humans can easily identify incoherent text, they have much more difficulty defining what makes a piece of text coherent. To make matters worse, multiple arrangements of segments might be equally coherent to a human. For extractive techniques, systems must devise an ordering of extracted segments and deal with “out-of-context” issues discussed above. For abstractive techniques, generation of fluent output from an abstract representation is sufficiently difficult that it is considered another sub-area in natural language

processing. Although output coherence is a requirement in both single- and multi-document summarization, the latter presents more problems (particularly for extractive systems) given the variety of sources extracts.

Overview of Selected Techniques

Due to relatively easy access to corpora, most research in summarization over the past two decades has been on written news. As most summarization systems today are primarily extractive, these methods will occupy the bulk of this discussion.

Extractive techniques first segment source text into smaller segments (sentences, paragraphs, etc.), which are then scored according a variety of features, e.g., position in the text [6], term and phrase frequencies [12], lexical chains (degree of lexical-connectedness between various segments) [1], topics present in the text [16], or discourse prominence [14]. A widely adopted approach is to use machine learning techniques to determine the relative importance of various features (the earliest example being [10]).

The features discussed above are relevant for both single- and multi-document summarization, although their relative importance varies with the task. Historically, the summarization field focused on the single-document case first, and then subsequently moved on to multi-document summarization. This move required systems to explicitly model similarities and differences in text to address redundancy, paraphrase, entailment, contradiction, and related linguistic issues. One general approach involves clustering, as exemplified by the MEAD framework [16]. Documents are first clustered to find topics present in the sources. Clusters are represented by their centroids, which are used to rank extracts (along with other features). Maximal Marginal Relevance (MMR) [7] is another effective algorithm, specifically designed for query-focused summaries (i.e., summaries that address an information need). It iteratively selects candidate segments to include in the final summary, balancing relevance and redundancy at each iteration. Redundancy is computed by content similarity between each candidate and the current summary state (using cosine similarity) – thus, candidates containing words already in the summary are penalized. Note that neither MEAD nor MMR explicitly deals with linguistic relationships such as paraphrase, but that issue has been specifically addressed in other work [8].

After scoring and selecting segments from source documents, extractive systems must decide on an ordering in the final system output. Ideally, the output should constitute a coherent piece of text. Simple baselines for ordering segments include extraction order (i.e., by score), temporal order (based on metadata or temporal expressions), and order in source document (preserving source structure). While simple to implement, these techniques frequently yield disfluent summaries. Coherence can be improved by applying computational models of content and discourse [2]. Nevertheless, text structuring is a relatively under-explored area of summarization, particularly due to difficulty in evaluation. As a final note, one possible alternative is to abandon the assumption of summaries as fluent prose, and instead present users with a bulleted list of extracts.

Although open-domain abstractive summarization using deep semantic representations is beyond the current state of the art, a variety of successful abstractive techniques operating on syntactic structures have been developed. Most of these techniques involve parsing source documents and manipulating the resulting parse trees. One popular approach involves “trimming”, or removing inessential structures from the parse tree [9,20] – for example, removing adjunct clauses that do not contribute much information. Other successful techniques include “splicing” fragments from multiple sentences (sometimes across multiple documents) – for example, embedding a simple sentence as a relative clause inside another [3,13]. Of course, these operations are not mutually exclusive. Syntactic manipulations are particularly helpful in multi-document summarization since sentences from different sources might partially overlap, e.g., a sentence contains both redundant and new information. In this case, syntactic operations can potentially deliver the best of both worlds, by eliminating redundant information and preserving new information. However, as Sparck Jones recently noted [19], there has been comparatively little work on abstractive summarization over the last decade.

Additional Readings

Beyond this entry, a number of additional sources are recommended for further reading: slides from a tutorial presentation at SIGIR 2004 [15] provide a good starting point. Special issues of the journal *Information Processing and Management* [19] and *Computational*

Linguistics [17] contain in-depth articles on selected topics. For details on specific summarization techniques, a good place to look is the online proceedings of the Document Understanding Conferences [4], an annual evaluation of summarization systems. A note on references in this entry: since a comprehensive bibliography is impossible due to space limitations, either representative early articles or recent ones are cited (in the latter case, the assumption is that the reader can trace citations backwards).

Key Applications

Summarization technology has a number of applications, many of which are outlined below:

Search result summarization. Search engines typically retrieve thousands of hits (if not more) in response to a user's query. Summarization systems can provide users with an overview of results to support information seeking.

Tools for analytical support. Summarization can be applied to support intelligence analysis, e.g., "prepare a report on recent insurgent activities in Basra", as well as similar activities such as investigative journalism and business intelligence.

Personal information agent. A personal information agent maintains a profile of the user's interest and proactively seeks out information (e.g., retrieving and summarizing relevant news items on a continuous basis).

Accessibility assistance. For example, a visually impaired person might make use of a screen reader augmented with summarization technology for greater efficiency.

Support for handheld devices. Handheld devices such as cell phones and PDAs with small screens could benefit from more condensed information.

Medical applications. Physicians struggle to keep current with the ever-increasing volume of medical literature. Summarization systems can be deployed to assist physicians, e.g., provide an overview of treatment options for a particular disease.

Summarization of meetings. Summarization technology can be coupled with speech recognizers to automatically generate "meeting minutes".

Future Directions

Current research in summarization can be characterized by three broad trends:

Increasing linguistic sophistication. Extractive techniques can benefit from richer features to characterize the appropriateness of a segment for inclusion in the summary – these features come from increasingly detailed linguistic analysis, enabled by advances in language processing technology. Of particular interest is the modeling of linguistic relations such as paraphrase, entailment, and contradiction. Separately, this task has been captured in the PASCAL recognizing textual entailment evaluations.

As discussed above, limitations of extractive methods can be addressed by incorporating abstractive techniques, e.g., manipulation of parse trees. Future developments appear to follow this trend, with increasingly richer representations (enabled by improvements in syntactic, semantic, discourse, and pragmatic analysis). In other words, abstractive summarization will likely be arrived at by successive approximations with hybrid techniques.

Exploration of different genres and domain-specific applications. Recently, researchers have become interested in "informal" text – a broad genre that includes emails, conversational speech, blogs, chat, SMS messages, etc. They are important because an increasing portion of our society's knowledge is captured in these channels. Furthermore, informal text push the frontiers of summarization technology by forcing researchers to develop more general and robust algorithms.

Integration with other language processing components. As technology matures, it becomes feasible to integrate summarization with other components to create more powerful applications. A few examples: integration with speech recognition to summarize TV broadcasts and meetings; integration with machine translation to summarize documents from multiple languages; integration with information retrieval and question answering to produce responses that answer complex questions.

Experimental Results

Summarization is fundamentally experimental in nature, as the effectiveness of different techniques cannot be derived from first principles. Thus, tools for assessing summary quality are critical to ensuring progress, and evaluation methods themselves represent an active area of research.

Methodologies for evaluating system output can be broadly classified into two categories: *intrinsic* and *extrinsic*. In an intrinsic evaluation, system output is

directly evaluated in terms of a set of norms – for example, fluency, coverage of key ideas, or similarity to an “ideal” summary (see [19] for an overview). In particular, the last criteria has been operationalized in ROUGE [11], a commonly used automated metric that compares system output to a number of human-generated “reference” summaries. In contrast, extrinsic evaluations attempt to measure how summarization impacts some other task, for example, helping users determine if a document is relevant (see [5] and references therein). While more informative, extrinsic evaluations are much more difficult to conduct, since it often involves constructing realistic scenarios for summarization systems.

One of the most important driving forces behind summarization research is the existence of annual evaluations that provide a community-wide benchmark to assess progress. Two such evaluations are the Document Understanding Conferences [4] sponsored by the U.S. National Institute of Standards and Technology (NIST), and the NTCIR Project sponsored by Japan’s National Institute of Informatics. Starting in 2008, DUC is replaced by the newly created Text Analysis Conference, also sponsored by NIST.

Data Sets

Instructions for obtaining data from the DUC and NTCIR evaluations can be found on their respective websites.

Cross-references

- ▶ [Information Extraction](#)
- ▶ [Information Retrieval](#)

Recommended Reading

1. Barzilay R. and Elhadad M. Using lexical chains for text summarization. In Proc. ACL/EACL Workshop on Intelligent Scalable Text Summarization, 1997.
2. Barzilay R. and Lee L. Catching the drift: Probabilistic content models, with applications to generation and summarization. In Proc. 2004 Human Language Technology Conf., 2004, pp. 113–120.
3. Barzilay R. and McKeown K.R. Sentence fusion for multidocument news summarization. *Computat. Linguist.*, 31(3):297–327, 2005.
4. Document Understanding Conferences. <http://duc.nist.gov/>.
5. Dorr B.J., Monz C., President S., Schwartz R., and Zajic D. A methodology for extrinsic evaluation of text summarization: Does ROUGE correlate? In Proc. ACL 2005 Workshop on Intrinsic and Extrinsic Evaluation Measures for MT and/or Summarization, 2005.

6. Edmundson H.P. New methods in automatic extracting. *J. ACM*, 16(2):264–285, 1969.
7. Goldstein J., Mittal V., Carbonell J., and Callan J. Creating and evaluating multi-document sentence extract summaries. In Proc. Int. Conf. on Information and Knowledge Management, 2000, pp. 165–172.
8. Hatzivassiloglou V., Klavans J.L., and Eskin E. Detecting text similarity over short passages: Exploring linguistic feature combinations via machine learning. In Proc. Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora, 1999.
9. Knight K. and Marcu D. Statistics-based summarization – step one: Sentence compression. In Proc. 12th National Conf. on AI, 2000, pp. 703–710.
10. Kupiec J., Pedersen J.O., and Chen F. A trainable document summarizer. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 68–73.
11. Lin C.Y. and Hovy E. Automatic evaluation of summaries using n-gram co-occurrence statistics. In Proc. 2003 Human Language Technology Conf., 2003, pp. 71–78.
12. Luhn H.P. The automatic creation of literature abstracts. *IBM J. Res. Develop.*, 2(2):159–165, 1958.
13. Mani I., Gates B., and Bloedorn E. Improving summaries by revising them. In Proc. 27th Annual Meeting of the Assoc. for Computational Linguistics, 1999, pp. 558–565.
14. Marcu D. The Rhetorical Parsing, Summarization, and Generation of Natural Language Texts. PhD Thesis, University of Toronto, 1997.
15. Radev D.R. Text summarization. In Tutorial Presentation at the 27th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004.
16. Radev D.R., Blair-Goldensohn S., and Zhang Z. Experiments in single and multi-document summarization using MEAD. In Proc. 2001 Document Understanding Conf., 2001.
17. Radev D.R., Hovy E., and McKeown K. Introduction to the special issue on summarization. *Computat. Linguist.*, 28(4):399–408, 2002.
18. Radev D.R. and McKeown K. Generating natural language summaries from multiple on-line sources. *Computat. Linguist.*, 24(3):469–500, 1998.
19. Sparck Jones K. Automatic summarising: The state of the art. *Inf. Process. Manage.*, 43(6):1449–1481, 2007.
20. Zajic D., Dorr B., Lin J., and Schwartz R. Multi-Candidate Reduction: Sentence compression as a tool for document summarization tasks. *Inf. Process. Manage.*, 43(6):1549–1570, 2007.

Summarization Correctness

- ▶ [Summarizability](#)

Summary

- ▶ [Data Sketch/Synopsis](#)

Supervised Learning

► Classification

Support Vector Machine

Hwanjo Yu

University of Iowa, Iowa City, IA, USA

Synonyms

SVM

Definition

Support vector machines (SVMs) represent a set of supervised learning techniques that create a function from training data. The training data usually consist of pairs of input objects (typically vectors) and desired outputs. The learned function can be used to predict the output of a new object. SVMs are typically used for classification where the function outputs one of finite classes. SVMs are also used for regression and preference learning, for which they are called support vector regression (SVR) and ranking SVM, respectively. SVMs belong to a family of generalized linear classifier where the classification (or boundary) function is a hyperplane in the feature space. Two special properties of SVMs are that SVMs achieve (i) high generalization (Generalization denotes the performance of the learned function on testing data or “unseen” data that are excluded in training.) by maximizing the margin (Margin denotes the distance between the hyperplane and the closest data vectors in the feature space.), and (ii) support efficient nonlinear classification by kernel trick (Kernel trick is a method for converting a linear classifier into a non-linear one by using a non-linear function to map the original observations into a higher-dimensional space; this makes a linear classification in the new space (or the feature space) equivalent to non-linear classification in the original space (or the input space).).

Historical Background

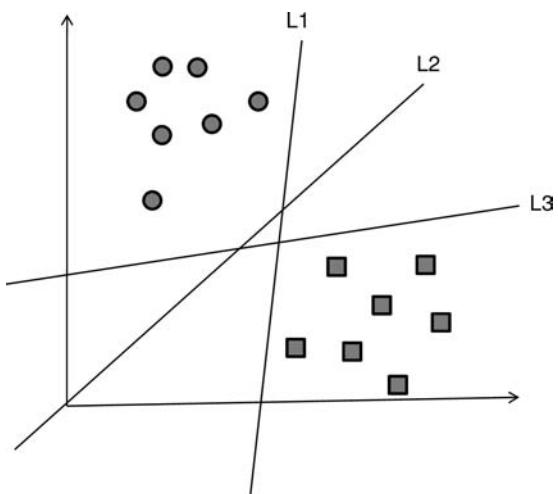
Vapnik developed the related concepts at 1979 and published an article in Russian that was translated to English at 1982. The first book introducing SVMs (written by him) were published at 1995. Since then, numerous literatures have been published including tutorials, journal articles, and books.

Foundations

SVMs were initially developed for classification [1] and have been extended for regression [4] and preference learning [3,5]. The initial form of SVMs are a binary classifier where the output of learned function is either positive or negative. A multiclass classification can be implemented by combining multiple binary classifiers using pairwise coupling method [2]. This section explains the motivation and formalization of SVM as a binary classifier, and the two key properties – margin maximization and kernel trick.

Motivation

Binary classification is to classify data objects into either positive or negative class. Each data object (or data point) is represented by a n -dimensional vector. Each of these data points belongs to only one of two classes. A *linear* classifier separates them with an “ n minus 1” dimensional hyperplane. For example, Figure 1 shows two groups of data and separating hyperplanes that are lines in a two-dimensional space. There are many linear classifiers that correctly classify (or divide) the two groups of data such as L1, L2 and L3 in Fig. 1. In order to achieve maximum separation between the two classes, An SVM pick the hyperplane so that the margin, or the distance from the hyperplane to the nearest data point, is maximized. Such a hyperplane is likely to generalize better, meaning that the hyperplane not only correctly classify the given or training data points, but also is likely to correctly classify “unseen” or testing data points.



Support Vector Machine. Figure 1. Linear classifiers (hyperplane) in two-dimensional spaces.

Formalization

The data points D in Fig. 1 (or training set) can be expressed mathematically as follows:

$$D = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \dots, (\vec{x}_m, y_m)\}, \quad (1)$$

where \vec{x}_i is a n -dimensional real vector, y_i is either 1 or -1 denoting the class to which the point \vec{x}_i belongs. The SVM classification function $F(\vec{x})$ takes the form

$$F(\vec{x}) \Rightarrow \vec{w} \cdot \vec{x} - b. \quad (2)$$

\vec{w} is the weight vector and b is the bias, which will be computed by SVM in the training process.

First, to correctly classify the training set, $F(\cdot)$ (or \vec{w} and b) must return positive numbers for positive data points and negative numbers otherwise, that is, for every point \vec{x}_i in D ,

$$\begin{aligned} \vec{w} \cdot \vec{x}_i - b &> 0 \text{ if } y_i = 1, \text{ and} \\ \vec{w} \cdot \vec{x}_i - b &< 0 \text{ if } y_i = -1 \end{aligned}$$

These conditions can be revised into:

$$y_i(\vec{w} \cdot \vec{x}_i - b) > 0, \forall (\vec{x}_i, y_i) \in D. \quad (3)$$

If such a linear function F that correctly classifies every point in D or satisfies (3) exists, D is called *linearly separable*.

Second, F (or the hyperplane) needs to maximize the *margin*. Margin is the distance from the hyperplane to the closest data points. An example of such hyperplane is illustrated in Fig. 2. To achieve this, (3) is revised into the following (4).

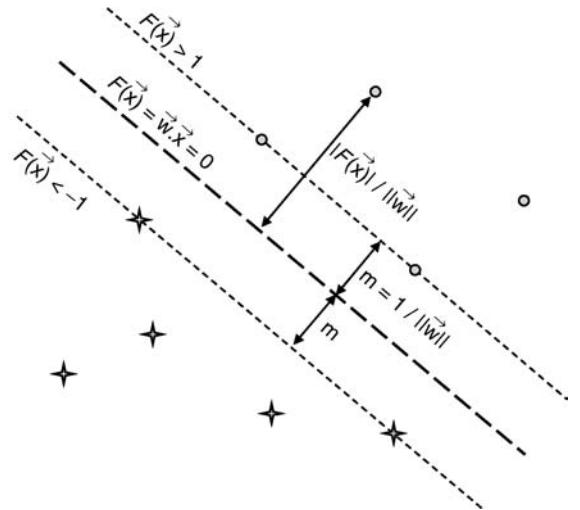
$$y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \forall (\vec{x}_i, y_i) \in D. \quad (4)$$

Note that (4) includes equality sign, and the right side becomes 1 instead of 0. If D is linearly separable, or every point in D satisfies (3), then there exists such a F that satisfies (4). It is because, if there exist such \vec{w} and b that satisfy (3), they can be always rescaled to satisfy (4).

The distance from the hyperplane to a vector \vec{x}_i is formulated as $\frac{|F(\vec{x}_i)|}{\|\vec{w}\|}$. Thus, the margin becomes

$$\text{margin} = \frac{1}{\|\vec{w}\|}. \quad (5)$$

because when \vec{x}_i are the closest vectors, $F(\vec{x})$ will return 1 according to (4). The closest vectors, that satisfy (4) with equality sign, are called *support vectors*.



Support Vector Machine. Figure 2. SVM classification function: the hyperplane maximizing the margin in a two-dimensional space.

Maximizing the margin becomes minimizing $\|\vec{w}\|$. Thus, the training problem in SVM becomes a constrained optimization problem as follows:

$$\text{minimize : } \frac{1}{2} \|\vec{w}\|^2, \quad (6)$$

$$\text{subject to : } y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1, \quad \forall (\vec{x}_i, y_i) \in D. \quad (7)$$

The factor of $\frac{1}{2}$ is used for mathematical convenience.

However, the optimization problem will not have a solution if D is not linearly separable. To deal with such cases, *soft margin* SVM allows mislabeled data points while still maximizing the margin. The method introduces slack variables, ξ , which measure the degree of misclassification. The following is the optimization problem for soft margin SVM.

$$\text{minimize : } L_1(w, b, \xi) = \frac{1}{2} \|\vec{w}\|^2 + C \sum_i \xi_i, \quad (8)$$

$$\begin{aligned} \text{subject to : } y_i(\vec{w} \cdot \vec{x}_i - b) &\geq 1 - \xi_i, \\ \forall (\vec{x}_i, y_i) &\in D. \end{aligned} \quad (9)$$

$$\xi \geq 0 \quad (10)$$

Due to the ξ in (9), data points are allowed to be misclassified, and the amount of misclassification or error will be minimized as well as the margin according to the objective function (8). C is a parameter that determines the tradeoff between the margin size and the amount of error in training.

This optimization problem is called quadratic programming (QP) problem, and it is the primal form of the QP. The primal form can be changed to the following dual form using the Lagrange multipliers.

$$\text{minimize: } L_2(\alpha) = \sum_i \alpha_i - \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\vec{x}_i, \vec{x}_j), \quad (11)$$

$$\text{subject to : } \sum_i \alpha_i y_i = 0 \quad (12)$$

$$C \geq \alpha \geq 0 \quad (13)$$

α constitute a dual representation for the weight vector such that

$$\vec{w} = \sum_i \alpha_i y_i \vec{x}_i. \quad (14)$$

$K(\vec{x}_i, \vec{x}_j)$ in (11) is originally a dot product of the two vectors, that is, $\vec{x}_i \cdot \vec{x}_j$. However, the dot product can be replaced by a non-linear kernel function, which allows the algorithm to fit the maximum-margin hyperplane in the transformed feature space. The transformed feature space is usually high dimensional, and the hyperplane (or linear classifier) in the high-dimensional space becomes non-linear in the original input space. Computing the kernel function K is often done as fast as computing a dot product. In this way, the complexity of computing a non-linear function becomes the same as that of computing a linear function in SVM. This method for computing a non-linear function is called *kernel trick*. The following are popularly used non-linear kernels.

1. Radial basis function (RBF): $K(a, b) = \exp(-\gamma \|a - b\|^2)$
2. Polynomial: $K(a, b) = (a \cdot b + 1)^d$
3. Sigmoid: $K(a, b) = \tanh(\kappa a \cdot b + c)$

Once α is learned from the dual form, the linear function F in (2) becomes the following non-linear function.

$$F(\vec{x}) \Rightarrow \vec{w} \cdot \vec{x} - b \Rightarrow \sum_i \alpha_i y_i K(\vec{x}_i, \vec{x}) - b \quad (15)$$

Key Applications

SVMs have been widely applied for object classification and pattern recognition such as text categorization, face detection in images, and handwritten digit recognition.

Cross-references

- ▶ Classification
- ▶ Regression

Recommended Reading

1. Burges C.J.C. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discovery*, 2:121–167, 1998.
2. Hastie T. and Tibshirani R. Classification by pairwise coupling. In *Advances in Neural Information Processing Systems*, 1998.
3. Herbrich R., Graepel T., and Obermayer K. (eds.) *Large margin rank boundaries for ordinal regression*. MIT Press, Cambridge, MA, 2000.
4. Smola A.J. and Scholkopf B. A tutorial on support vector regression. Technical Report, NeuroCOLT2 Technical Report NC2-TR-1998-030, 1998.
5. Yu H. SVM selective sampling for ranking with application to data retrieval. In Proc. 11th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2005.

Supporting Transaction Time Databases

DAVID LOMET

Microsoft Research, Redmond, WA, USA

Synonyms

Temporal database; Multi-version database

Definition

The temporal concepts glossary maintained at <http://www.cs.aau.dk/~csj/Glossary/> defines transaction time as: “The *transaction time* of a database fact is the time when the fact is current in the database and may be retrieved.” A transaction time database thus stores versions of database records or tuples, each of which has a start time and an end time, delimiting the time range during which they represent the current versions of database facts. As each version is the result of transactions, the times associated with the version are the times for the transaction starting the version (the start time) and for the transaction ending the version (the end time). These transaction times are required to agree with the serialization order of the transaction, so that the database can present a transaction consistent view of the facts being stored.

Historical Background

Postgres was the first database system that supported transaction time databases [13]. It implemented a

prototype relational database system using a versioning approach for recovery. This was then augmented with a version store based on the R-tree [4]. Subsequently, the TSB-tree was introduced [7], based on the WOB-tree time splitting [3], which provided an integrated approach to storing both current and historical data.

Commercially, Oracle and Rdb [5] database systems both support multi-version concurrency control, now called snapshot isolation. Oracle subsequently added a Flashback [11] feature that permitted access to historical versions, based on saving a linear history of their recovery versions. While this permits access to historical versions, it is mostly intended as an efficient means of providing an online backup for “point-in-time” recovery, a form of “media” recovery in which the database is recovered to a point just preceding a bad user transaction.

Temporal databases have been extensively studied [14], including not only transaction time but also valid time. Included as well are bi-temporal databases, where both valid and transaction time are supported. This work has clarified the conceptual issues and provided a common vocabulary of terms for describing work in the field.

Foundations

Transaction time databases usually offer the full range of database functionality that one would expect of a database storing only current facts, which is called a current time database. In addition, a transaction time database provides access to all prior facts as represented by versions of records that existed at some prior time. Several types of functionality can be envisioned.

1. Access to database facts “as of” some past time, which are called “as of” queries
2. Access to versions of a database fact in some time range, which are called “time travel” queries
3. Access to collections of database facts in some time range, i.e., “general transaction time queries”

The functionality provided by transaction time databases is important for applications such as time series data, regulatory compliance, repeatability of scientific experiments, etc. Further, a transaction time database can provide valuable system capabilities such as snapshot isolation concurrency control, recovery from bad user transactions, and recovery from media failure.

To support transaction time functionality, one needs to change the semantics of the data manipulation operations “update” and “delete.” An update creates an additional version instead of overwriting the prior state, retaining both new and old versions of the data. A delete is strictly logical, providing an end time for the previously current version. In this way, the prior version persists so that queries about the database while this version was alive can be answered.

Database systems supporting transaction time data have been built that are based on the relational data model, though this is not a fundamental limitation, simply a pragmatic choice.

Implementation Approaches

There are two generic approaches to implementing transaction time databases, which are described here.

Layered Approach The premise of the layered approach is that application programmers cannot wait for vendors to build transaction time functionality into database products [15]. Rather, a middleware layer (MWL) is implemented that provides this functionality. The MWL processes data definition statements, adding timestamp fields to each record, processes data manipulation statements (queries plus updates) written in a language that exposes temporal functionality and translates them into equivalent ordinary SQL queries. Typically, start time and end time are added to each record of a transaction time table. The table may be organized by a clustering key that includes (user defined primary key, start time, end time). Thus, a record with a given user defined primary key is clustered next to its earlier versions, which makes “time travel” queries efficient, while implying that “as of” queries will have relatively poor performance. Grouping by time does not help much with “as of” performance, and usually compromises “time travel” performance.

Built-In Approach The built-in approach requires the ability to modify the database engine to provide transaction time support. While a significant barrier, building transaction time support into a database system can greatly improve performance, bringing it close to current time database performance. This improved performance is the result of optimizations that are possible (i) for update, when the timestamps need to be added to versions, (ii) for storage with simple forms

of version compression, and (iii) for query, because specialized indexing is possible that improves data clustering. The rest of this article discusses the issues of built-in support and how to make this support perform well.

Managing Versions

Transaction time functionality requires dealing with the multiple versions of database facts that exist to express the states of the database over time. The built-in approach has more freedom than the MWL approach in how to achieve efficiency and performance. There are a number of issues, the more important ones being:

Timestamps Each historical version stored in a transaction time database has a begin time, at which it first became the current version, and an end time, at which it was either replaced by another version or deleted. Current versions have the usual start time, but have a special end time called “now” [2]. An MWL approach usually includes both start and end times with each version so that the SQL queries resulting from translating temporal queries are simple and efficient. With built-in support, most systems [6,13] store only the start time with each version, the end time being derived from the start time of the subsequent version. For a deletion, this next version can be a special “delete stub” version. For a query “as of” time T, the system then looks for the version of each record with the largest start time $\leq T$.

Storing Versions on a Page The common approach for organizing pages for current time databases is called a slotted array. Each array element points to a record on the page. For B-trees, these records are maintained in B-tree key order. When adding temporal support to a current time database system, it is convenient to minimize the change required for current time functionality. This argues for retaining the slotted array, and back-linking versions where each version is augmented with a pointer to its preceding version. Then for each record accessed in a query, the system follows this backward chain to the first version with a timestamp $\leq T$, the “as of” time requested.

Indexing Versions Being able to index historical versions by time is essential to avoid increasing costs for ever earlier query times [12]. Postgres [13] used the R-tree [4] for this. The TSB-tree [7] is a more specialized

index that can, with an appropriate page splitting policy [1] provide guarantees about the performance of “as of” queries. Its special feature is the introduction of a time split [3] where the time interval of a full database page is partitioned, with record versions being assigned to the resulting pages whenever their lifetimes intersect the time interval of a resulting page. Thus, a version whose lifetime intersects both resulting pages will be replicated in both pages. The result is that, when combined with ordinary B-tree key splitting, each TSB-tree page contains all versions within a key-time rectangle of the search space. This enables identifying exactly which pages can contain answers to a temporal query. Despite the need for replicating versions, the space required for the versions remains linear in the number of unique versions.

Compressing Versions The way that versions are stored on a page and indexed makes compressing versions simple. Usually, an update changes only a small part of a record, perhaps only a single attribute. Thus, delta compression, where the compressed version represents the difference between one version and another that is adjacent in time order, can be very effective. Only the updated attribute together with location information and timestamp needs to appear in the compressed version. Backward delta’s are to be preferred because this leaves the current time data uncompressed and hence unchanged, important both for compatibility and current time performance. Because time splits in the TSB-tree always replicate versions spanning the split time, and because splitting at current time is convenient, the last version in each page is always uncompressed, and this is preserved during a time split. Decompressing a version never needs information from any other page than the page upon which the version is stored.

Dealing with Timestamps

In a transaction time database, for a record identified in some way, its versions are distinguished by timestamps. The nature of timestamps, when they are chosen and included, how to optimize this process, and how to deal with user requests for transaction time are discussed below.

Nature of Timestamps Several forms of timestamp have been used for temporal support. Some systems use transaction identifiers (XIDs) instead of time, sometimes maintaining a separate table that maps XID to

time. When versioning is limited, e.g., to only support multiversion concurrency control, an active transaction's XID may be mapped to a list of transactions (their XIDs) that committed before them, hence determining which transactions have updates that should be visible to the active transaction [5]. However, for more general functionality, system time is usually used. It may need to be augmented with a sequence number because its granularity may not be sufficient to completely distinguish every transaction's updates.

When to Timestamp A timestamp for a version must enable “as of” queries to always see a transaction consistent view of the data. This can be achieved when timestamp order agrees with the serialization order of transactions. If one chooses timestamps prior to updating, the timestamp can be added immediately to versions generated by the updating. However, early timestamp choice means that transactions that serialize differently must be aborted. Most implementations of transaction time functionality thus choose timestamps at commit, where the commit order is the same as the serialization order for the transactions. This means, however, that the timestamp is not available at time of update, and must be added later.

Lazy Timestamping When a transaction's timestamp is determined late in the transaction, e.g., at commit time, preceding updated records need to be revisited to add the timestamp. Typically, an XID is placed in an updated record at update time, to be replaced later by the system time. Eager timestamping replaces XID with time prior to transaction commit, logging this activity as another update. This can be costly, so a lazy approach is generally preferred in which XID is replaced by time after the transaction commits. The mapping from XID to system time must be maintained persistently, at least until the timestamping is complete, to ensure that replacing XID with time can continue after a possible system crash.

Impact of User Requested Time The SQL language supports a user's request for current (transaction) time within a query. It is essential that the user see a time that is consistent with the transaction timestamp used for updates of the transaction. Providing the user with a time for the transaction while the transaction is executing constrains the choice of timestamp when the transaction is committed [9]. A transaction must be aborted if a timestamp cannot be chosen that is consistent with the

time provided to the user. To provide this, the system can exploit the fact that the user time request is usually not for the full precision of the timestamp, e.g., SQL DATE constrains only the date part of the timestamp. Further, remembering the largest timestamp on data that is seen by the transaction provides a lower bound for a possibly non-empty interval for timestamp choice.

Additional Uses

The versions maintained by a transaction time database can support a variety of other system uses.

Snapshot Isolation Recent versions can be used to provide snapshot isolation. With snapshot isolation (the default concurrency provided by Oracle), a transaction reads not the current data (which would be used for a serializable transaction) but a snapshot (version) current as of the start of the transaction or as of the first data read by the transaction. A transaction time database keeps these versions as well as possibly older versions. Thus, efficiently providing transaction time support also provides efficient snapshot isolation. The system may choose to garbage collect older versions more quickly when they are only used for concurrency control purposes.

Online Backup A database backup is simply an earlier state of the database. Transaction time databases make all earlier states accessible and queryable. To use an earlier transaction time database state as a backup for, e.g., media recovery for the current state, requires two things: (i) the earlier state needs to be on a separate medium than the current state; and (ii) the media recovery log needs to include all updates from the earlier state forward to the current state and itself be on a separate device. A transaction time database system can use time splits (which it would use in a TSB-tree) to move versions to separate backup media, and it can do this incrementally as well [8].

Bad User Transactions Occasionally, erroneous transactions commit, compromising the correctness of a database. Point-in-time recovery, where the database state is reset to an earlier time, just prior to the bad transaction, is the usual way of dealing with this. Conventional database backups used for this incur a long restore time followed by a roll forward to the just earlier time. A transaction time database lends itself greatly shortening the outage caused by this problem because earlier versions of the database are already maintained online.

Oracle Flashback [11] implements point-in-time recovery in this way. One can further limit the outage by identifying exactly which transactions should be removed from the database by tracking transaction read dependencies [10].

Key Applications

In addition to the system uses just described, transaction time databases are valuable for several applications, e.g., time series analysis, repeating experiments or analysis on historical data, and auditing and legal compliance.

Future Directions

Data stream processing is a new functionality that has several important applications requiring fast reaction to sequences of events, e.g., stock market data. This data may also be stored and more carefully analyzed. It is quite natural to think about stream data as transaction time data, and ask temporal queries of it.

Cross-references

- ▶ Concurrency Control
- ▶ Multi-Version Database
- ▶ Temporal Database
- ▶ Temporal Strata
- ▶ Transaction
- ▶ Transaction-Time Indexing

Recommended Reading

1. Becker B., Gschwind S., Ohler T., Seeger B., and Widmayer P. An asymptotically optimal multiversion B-tree. VLDB J., 5(4):264–275, 1996.
2. Clifford J., Dyreson C., Isakowitz T., Jensen C.S., and Snodgrass R.T. “On the semantics of “now” in databases,” ACM Trans. Database Syst., 22(2):171–214, 1997.
3. Easton M. Key-sequence data sets on inedible storage. IBM J. Res. Dev., 30(3):230–241, 1986.
4. Guttman A. R-trees: a dynamic index structure for spatial searching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984, pp. 47–57.
5. Hobbs L. and England K. Rdb: A Comprehensive Guide. Digital, 1995.
6. Lomet D.B., Barga R., Mokbel M., Shegalov G., Wang R., and Zhu Y. Transaction time support inside a database engine. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 35.
7. Lomet D.B. and Salzberg B. Access methods for multiversion data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1989, pp. 315–324.
8. Lomet D.B. and Salzberg B. Exploiting a history database for backup. In Proc. 19th Int. Conf. on Very Large Data Bases, 1993, pp. 380–390.

9. Lomet D.B., Snodgrass R.T., and Jensen C.S. Using the lock manager to choose timestamps. In Proc. Int. Conf. on Database Eng. and Applications, 2005, pp. 357–368.
10. Lomet D.B., Vagena Z., and Barga R. Recovery from “bad” user transactions. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 337–346.
11. Oracle. Oracle Flashback Technology (2005) http://www.oracle.com/technology/deploy/availability/htdocs/Flashback_Overview.htm
12. Salzberg B. and Tsotras V.J. A comparison of access methods for time-evolving data. ACM Comput. Surv., 31(2):158–221, 1999.
13. Stonebraker M. The design of the POSTGRES storage system. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 289–300.
14. Tansel U., Clifford J., Gadia S.K., Segev A., and Snodgrass R.T. Temporal databases: theory, design, and implementation. Benjamin/Cummings, 1993.
15. Torp K., Snodgrass R.T., and Jensen C.S. Effective timestamping in databases. VLDB J., 8(4):267–288, 2000.

Surfing

- ▶ Browsing in Digital Libraries

SVD Transformation

- ▶ Singular Value Decomposition

SVM

- ▶ Support Vector Machine

Switch

- ▶ OR-Split

Symbol Graph

- ▶ Symbolic Representation

Symbol Plot

- ▶ Symbolic Representation

Symbolic Graphic

- ▶ Symbolic Representation

Symbolic Representation

HANS HINTERBERGER
ETH Zurich, Zurich, Switzerland

Synonyms

Symbolic graphic; Symbol graph; Symbol plot

Definition

A written character or mark used to represent something; a letter, figure, or sign conventionally standing for some object, process etc. (Oxford English Dictionary). Examples are the figures denoting the planets, signs of the zodiac, etc. in astronomy; the letters and other characters denoting elements, etc. in chemistry, quantities, operations, etc. in mathematics, the faces of a crystal in crystallography.

In data visualization, the use of symbols allows the representation of multivariate data items, where each variate contributes to the symbol. The set of symbols may be displayed in an array, superimposed on coordinates to put extra information on a point plot or, if appropriate, on a geographical map.

When symbolic representations of information is used as a tool for thought or a form of communication, one distinguishes between abstract symbols where the graphical units are shapes formed by lines, and areas and depictive symbols (pictograms) where the graphical units are pictorial representations of objects and scenes.

Key Points

Symbols have considerable potential as an aid to support human cognition and as a medium for communication. To what extent symbols can successfully convey specific information and what their inherent limitations as a medium of communication are is far from being understood. There is nevertheless a useful and accumulating body of research dealing with the systematic application of symbolic representations.

One of the earliest researchers to treat symbolic representations in a theoretical context is the French cartographer Jacques Bertin. With his *Semiology of Graphics* [1] he organized the visual and perceptual elements of graphics according to the features and relations in data, distinguishing primarily between diagrams, networks, maps and symbols.

Among the first efforts to relate visual and perceptual research to the practical problems of designing information displays was the NATO Conference on Visual Presentation of Information in 1978. The contributions to this conference are published in [2], a book that deals with basic psychological issues as well as methods of evaluation of information design, with much room given to the use of symbols.

For a classification of symbols as used for data visualization and an overview for what type of functions different symbols are used the reader is referred to [3]. Tufte, in his series of books on information visualization, discusses with exemplary images and diagrams a wide range of different styles and techniques, good and bad, for the use of symbols, [4] is a starting point.

Cross-references

- ▶ Data Visualization
- ▶ Graph
- ▶ Table
- ▶ Thematic Map

Recommended Reading

1. Bertin J. *Semiology of Graphics* (translation by W.J. Berg), University of Wisconsin Press, Madison, WI, 1983.
2. Easterby R. and Zwaga H. (eds.). *Information Design, The Design and Evaluation of Signs and Printed Material*, Wiley, London, 1984.
3. Harris R.L. *Information Graphics: A Comprehensive Illustrated Reference*, Oxford University Press, New York, 1999.
4. Tufte E.R. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, 1983.

Symmetric Encryption

NINGHUI LI
Purdue University, West Lafayette, IN, USA

Synonyms

Secret-key encryption

Definition

Symmetric encryption, also known as secret key encryption, is a form of data encryption where a single secret key is used for both encryption and decryption.

Key Points

Modern symmetric encryption algorithms are often classified into stream ciphers and block ciphers. In a stream cipher, the key is used to generate a pseudo-random key stream, and the ciphertext is computed by using a simple operation (e.g., bit XOR or modular addition) to combine the plaintext bits and the key stream bits. Many stream ciphers implemented in hardware are constructed using linear feedback shift registers (LFSRs). The use of LFSRs on their own, however, is insufficient to provide good security. Additional variation and enhancement are needed to increase the security of LFSRs. RC4 is the most widely-used software stream cipher and is used in popular protocols such as Secure Sockets Layer (SSL) (to protect Internet traffic) and WEP (to secure wireless networks).

A block cipher operates on large blocks of digits with a fixed, unvarying transformation. The Data Encryption Standard (DES) [1] algorithm uses blocks of 64 bits; the Advanced Encryption Standard (AES) [2] algorithm uses 128-bit blocks. When using a block cipher to encrypt a message, one needs to choose an encryption mode. Commonly used modes include the Electronic Codebook (ECB), Cipher-block chaining (CBC), Cipher feedback (CFB), Output feedback (OFB), and Counter (CTR).

Cross-references

- ▶ Asymmetric Encryption
- ▶ Data Encryption

Recommended Reading

1. Federal information processing standards publication 46-3: Data encryption standard (DES), 1999.
2. Federal information processing standards publication 197: Advanced encryption standard, November 2001.

Synchronization Component

- ▶ Concurrency Control Manager

Synchronization Join

- ▶ Workflow Join

Synchronizing Distributed Transactions

- ▶ Distributed Concurrency Control

Synchronous Join

- ▶ OR-Join

Synchronous Pipelines

- ▶ Iterator

Synopsis

- ▶ Structure Indexing
- ▶ Synopsis Structure

Synopsis Structure

PHILLIP B. GIBBONS
Intel Labs Pittsburgh, Pittsburgh, PA, USA

Synonyms

Synopsis

Definition

A *synopsis structure* for a dataset S is any summary of S whose size is substantively smaller than S . Formally, its size is at most $O(|S|^\varepsilon)$, where $|S|$ is the size (in bytes) of S , for some constant $\varepsilon < 1$.

Key Points

Synopsis structures are small, often statistical summaries of a data set. The term serves as an umbrella for any summarization structure of sufficiently small size, such as random samples, histograms, wavelets, sketches, top-k summaries, etc.

Synopsis structures are most commonly used in conjunction with data streams. The goal is to construct, in one pass over the data stream, a synopsis structure that can be used to answer any query from a prespecified class of queries. That is, at any point, a user may pose a query

Q on the data stream thus far, and a (typically approximate) answer to Q must be produced using only the current synopsis structure. Two key advantages of using a synopsis structure to answer queries are that the space overhead is low (a massive dataset can be summarized using only a small amount of space) and the response time is fast (e.g., disk accesses can be avoided altogether). Moreover, in the common setting of queries comparing or aggregating over a distributed collection of streams, only the small synopsis structures (and not the massive data streams) need to be communicated between the collection points in order to answer the query.

Synopsis structures are also used within relational and XML databases, both for query optimization and for approximate query answering.

Common metrics for evaluating a synopsis structure include (i) *Coverage*: the range and importance of the class of queries supported; (ii) *Answer quality*: the accuracy and confidence of its (approximate) answers; (iii) *Space footprint*: its size, where smaller is better and often polylog space is desired (i.e., space that is $O(\log^k(|S|))$ for some constant $k \geq 1$); (iv) *Per-item processing time*: the total time to process the dataset, normalized to the number of items in the dataset; and (v) *Query time*: the time to answer a query from the synopsis structure [1].

Sometimes *sketch* is used interchangeably with synopsis structure, but more typically “sketch” is restricted to synopses based on random projections.

The term was coined by Gibbons and Matias in 1995.

Cross-references

- ▶ [Data Stream](#)
- ▶ [Histogram](#)
- ▶ [Random Sample](#)
- ▶ [Sketch](#)
- ▶ [Wavelets on Streams](#)

Recommended Reading

1. Gibbons P.B. and Matias Y. Synopsis data structures for massive data sets. DIMACS Series in Discrete Mathematics and Theoretical Computer Science: External Memory Algorithms, 1999.

Synthetic Image

- ▶ [Image](#)

Synthetic Microdata

JOSEP DOMINGO-FERRER

Universitat Revira i Virgili, Tarragona, Catalonia

Synonyms

[Imputed data](#); [Simulated data](#); [Multiple imputation](#)

Definition

Publication of synthetic – i.e., simulated – data is an alternative to masking for statistical disclosure control of microdata. The idea is to randomly generate data with the constraint that certain statistics or internal relationships of the original dataset should be preserved.

Key Points

The operation of the original proposal by Rubin [2] is next outlined. Consider an original microdata set X of size n records drawn from a much larger population of N individuals, where there are background attributes A , non-confidential attributes B and confidential attributes C . Background attributes are observed and available for all N individuals in the population, whereas B and C are only available for the n records in the sample X . The first step is to construct from X a multiply-imputed population of N individuals. This population consists of the n records in X and M (the number of multiple imputations, typically between 3 and 10) matrices of (B,C) data for the $N - n$ non-sampled individuals. The variability in the imputed values ensures, theoretically, that valid inferences can be obtained on the multiply-imputed population. A model for predicting (B,C) from A is used to multiply-impute (B,C) in the population. The choice of the model is a non-trivial matter. Once the multiply-imputed population is available, a sample Z of n' records can be drawn from it whose structure looks like the one of a sample of n' records drawn from the original population. This can be done M times to create M replicates of (B,C) values. The results are M multiply-imputed synthetic datasets. To make sure no original data are in the synthetic datasets, it is wise to draw the samples from the

Synthetic Data

- ▶ [Matrix Masking](#)

multiply-imputed population excluding the n original records from it.

There are other approaches to synthetic data generation based on bootstrap, Latin Hypercube sampling, Cholesky decomposition, etc. More information can be found in [1].

Cross-references

- ▶ [Inference Control in Statistical Databases](#)
- ▶ [Microdata](#)

Recommended Reading

1. Hundepool A., Domingo-Ferrer J., Franconi L., Giessing S., Lenz R., Longhurst J., Nordholt E.S., Seri G., and De Wolf P.-P. Handbook on Statistical Disclosure Control. CENEX SDC Project, November 2006 (manuscript version 1.0). <http://neon.vb.cbs.nl/CENEX/>.
2. Rubin D.B. Discussion of statistical disclosure limitation. *J. Off. Stat.*, 9(2):461–468, 1993.

System Catalog

- ▶ [Data Dictionary](#)

System R (R*) Optimizer

MOUNA KACIMI, THOMAS NEUMANN
Max-Planck Institute for Informatics, Saarbrücken,
Germany

Definition

The System R Optimizer is the cost-based query optimizer of System R. It pioneered several optimization techniques, including using dynamic programming for bottom-up join tree construction, and the concept of interesting orderings for exploiting ordering in intermediate results. Later, it was generalized for distributed database systems in System R*.

Historical Background

System R is a database management system based on a relational data model that was proposed by E. F. Codd [4] in 1970. The system offers data independence by providing a high-level user interface through which the end user deals with data content rather than the underlying storage structures. In other words, users do not need to know how the tuples are physically stored

and which access paths are available to write queries. Thus, data storage structures may change over time without users being aware of it, providing a high level of data independence and user productivity. Moreover, System R offers capabilities for database management in realistic and operational environments. Particularly, it supports multiple users concurrently accessing data, provides means for system recovery after hardware or software failures, supports different types of database use including ad hoc queries, programmed transactions and report generation. System R has been developed at the San Jose IBM Research Laboratory during three phases. First, *Phase Zero* of the project started in 1974 and ended in 1975. It involved the development of a high-level relational user language, called SQL. During this phase, a subset of SQL was implemented for one user at a time. One of the most challenging tasks of Phase Zero was the design of optimizer algorithms for efficient query processing. Second, *Phase One* took place from 1976 to 1977. It involved the development of a full-function multi-user version of System R. The multi-user prototype contained new subsystems, such as locking subsystems that prevent conflict of concurrent user accesses. Finally, *Phase Two* focused on evaluating the System R during 1978 and 1979. It was mainly composed of two parts: (i) evaluation of the system at the San Jose research Laboratory, and (ii) evaluation of the actual use of the system at a number of internal sites of IBM.

Foundations

This section is organized as follows. First, it starts by describing the role of the optimizer in processing SQL statements. Then, it describes the storage components used to access paths on the different relations. Next, it presents optimization techniques for single relations and joins in System R. Finally, it describes how these techniques are generalized to the distributed case of System R*.

SQL Query Processing

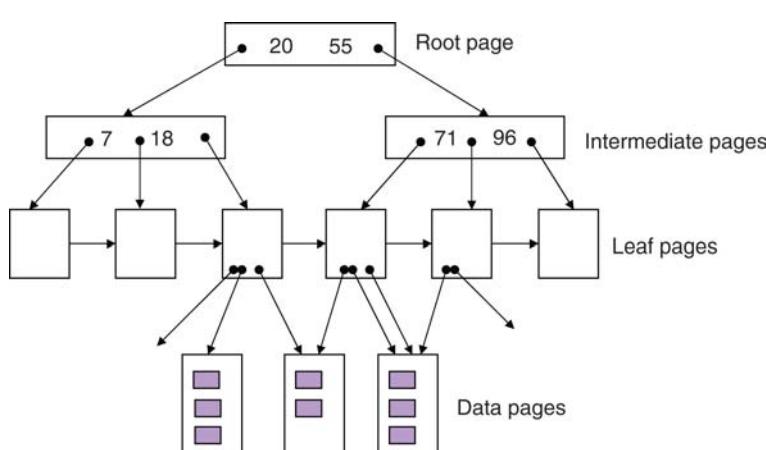
An SQL statement is composed of one or multiple query blocks depending on whether the operands of the used predicates are simple values (of the form “*column operator value*”) or queries (of the form “*column operator query*”). A *query block* is represented by: (i) a *SELECT* list containing the list of items to be retrieved, (ii) a *FROM* list containing the relation(s)

references and (iii) a *WHERE* tree containing the boolean combination of simple predicates specified by the user. Processing SQL statements requires four main phases namely: parsing, optimization, code generation and execution. During the first phase, each SQL statement is sent to the *Parser* to check its syntax. If no errors are detected, the optimizer component is called in the second phase. Using the System R catalogs, the optimizer verifies the existence of all the relations and columns referenced, and collects information about them. It gets from the catalog the datatype and the length of each column, and use these information to check the semantic errors and type compatibility in both expressions and predicate comparisons of the SQL statement. In addition, the optimizer obtains, from the catalogs, statistics about the referenced relations and the access paths available on each of them and use these information for access path selection process. Then, the optimizer chooses a query plan, from a tree of alternate path choices, that is a minimum cost solution. This chosen plan is a tree represented in the Access Specification Language (ASL) [7]. In the third phase, the Code generator translates ASL trees into machine language code to finally execute the plan chosen by the optimizer in the fourth phase. When the code is executed, it accesses the Relational Storage System (RSS), via the Storage System Interface (RSI), to scan each of the query relations using the access paths chosen by the optimizer. Even though the RSS may be used for different purposes, here, we focus on its use for computing cost formulas and executing the code generated by the query processing in System R.

Relational Storage System

The Relational Storage System (RSS) provides underlying storage support for System R. Relations in the RSS are stored as a collection of tuples whose columns are physically contiguous. The storage space is logically organized into segments. Each relation resides within a single segment, however, a segment may contain one or more relations. A segment is composed of a set of equal-sized pages. Each tuple of a relation is stored within a single page and is assigned an identification of the relation to which it belongs. Note that a page may contain tuples from one or more relations.

To access tuples in a relation, an RSS scan is used. Along a given access path, a scan returns one tuple at a time. Two different types of scans can be distinguished. First, *Segment* scans find all the tuples of a given relation by examining the segment that contains the relation. All the non-empty pages of the segment are touched and the tuples belonging to the given relation are returned. Second, *Index* scans access a relation in value order using an index. An index is created on one or more columns of a relation. It is composed of one or more pages within the segment containing the relation. The pages of the index are separated from the pages containing the relation tuples. They are organized into a B-tree structure as shown in Fig. 1. Each page is a node of the B-tree that contains an ordered sequence of index entries. An entry of a non-leaf node consists of a $\prec key, pointer \succ$ pair, where the pointer address another page in the same tree. Leaf pages contain sets of $\prec key, identifier \succ$ pairs, where identifier indicates the tuple that contains the corresponding key. An



System R (R*) Optimizer. Figure 1. Example of B-tree index structure.

index scan does a sequential read along the leaf pages to get the tuple identifiers matching a given key. These identifiers are used to find and return the data tuples to the user in key value order.

When an index scan examines a relation, each page of the index is touched only once, but a data page may be touched more than once. This case may happen if a page contains tuples which are not close in the index ordering. An index is said to be clustered when the physical proximity of tuples in the same data page corresponds to the index key value. Therefore, not only each index page, but also each data page will be touched only once in a scan on that index. Additionally, to reduce the number of touched pages, starting and stopping key values can be specified when scanning tuples. Thus, only the tuples whose keys belong to the predefined interval are returned. Both index and segment scans may use a set of predicates, called also search arguments or SARGS. These predicates are of the form (“*column operator value*”). They are applied to tuples before they are returned to the RSI caller. Sargable predicates play an important role in reducing the cost by eliminating unnecessary RSI calls for tuples which can be rejected within the RSS.

Access Path Selection for Single Relations

This section describes how the optimizer chooses a query plan accessing a single relation. As presented previously, the optimizer gets from the System R catalog the access paths available on the relation referenced by the query. The cheapest access path is obtained by evaluating the cost for each available access path, i.e., each index on the relation plus a segment scan. The optimizer formulates a cost prediction given by the following formula:

$$\text{Cost} = \text{Page Fetches} + W \times (\text{RSI calls})$$

where *Page Fetches* represent *I/O* requirement computed by the number of index pages fetched plus number of data pages fetched. *RSI calls* indicate the predicted number of tuples returned from the RSS. This number is a good approximation of *CPU* utilization since most of System R’s CPU time is spent in RSS. The parameter *W* is an adjustable weighting factor between *I/O* and *CPU*.

To find the cheapest access plan for a single relation query, the optimizer needs to examine the cheapest *unordered* access path and the cheapest access path producing tuples in each *interesting* order. *Unordered* access path may produce tuples in some order, but

the order is not *interesting*. In this case, the optimizer simply chooses the cheapest access path as query plan. By contrast, a tuple order is an *interesting order* if that order is specified by the query block using GROUP BY or ORDER BY clauses. In this case, the optimizer compares the cost of producing that interesting order to the cost of the cheapest unordered path plus the cost of sorting QCARD tuples into the proper order, where QCARD represents the query cardinality. The cheapest of these alternatives is chosen as the plan for the query block. In the following, a description is given of how the query cardinality (QCARD) and RSI calls are computed by the query optimizer.

During the query processing, the optimizer gets statistics on the relations of the query and access paths available on each relation. These statistics include the cardinality of the query relations, the number of segments holding the relevant relations and the fraction of their contained pages, the number of distinct keys and the number of pages of each index. The statistics are used by the optimizer to assign a *selectivity factor* *F* to each predicate of the WHERE tree. This selectivity factor indicates the expected fraction of tuples which will satisfy the predicate. More details on statistics and selectivity factors are given in [8]. The optimizer computes the query cardinality (QCARD) as the product of the cardinalities of every relation in the query block’s FROM list times the product of all the selectivity factors of that query block’s predicates. The number of RSI calls (RSICARD) is the product of the relation cardinalities times the selectivity factors of the sargable predicates, since the sargable predicates filter out tuples without returning across the RSS interface.

Access Path Selection for Joins

A join query in SQL combines tuples from more than one relation. Two join methods have been identified as optimal or nearly optimal in most cases. For simplicity, a description of how to join two relations is given, then, it is extended to *n* relations. A two-way join involves two relations respectively called *outer* relation and *inner* relation. A predicate that relates columns of two relations to be joined is called *join predicate*. The columns referenced in a join predicate are called *join columns*. Consider the following example:

```
SELECT Name, Location
FROM STUDENT, DEPARTMENT
WHERE STUDENT.Department=DEPARTMENT.Num
```

In this example, the outer and inner relations are respectively “STUDENT” and “DEPARTMENT”. There is one join predicate that is “STUDENT.Department = DEPARTMENT.Num.” The join columns of this query are “STUDENT.Department” and “DEPARTMENT.Num”.

The first supported join method is called *nested loops*. This method scans the outer and the inner relations in any order. The scan on the outer relation is opened and for each outer tuple obtained, a scan is opened on the inner relation to retrieve, one at a time, all the tuples of the inner relation that satisfy the join predicate. The cost of a nested loop join is computed from the costs of scans on single relations defined in the previous section and is given by the following formula:

$$\begin{aligned} \text{C-nested-loop-join (path1, path2)} \\ = \text{C-outer(path1)} + N \times \text{C-innter(path2)} \end{aligned}$$

where $C\text{-outer}(path1)$ indicates the cost of scanning the outer relation via path1, $C\text{-inner}(path2)$ indicates the cost of scanning the inner relation, applying all applicable predicates, and N is the (product of the cardinalities of all relations R of the join so far) \times (product of the selectivity factors of all applicable predicates).

The second supported join method is called *merging scans*. It requires the outer and the inner relations to be scanned in join column order. This means that join columns define *interesting orders* in addition of columns mentioned in GROUP BY and ORDER BY clauses. In case the join query contains more than one predicate, one of them is used as join predicate and the others are treated as ordinary predicates. If a relation has no index on the join column, it has to be sorted into a temporary list ordered by join column. By using ordering on join columns, the merging scan method avoids rescanning the entire inner relation for each tuple of the outer relation. The merging scan synchronizes the inner and the outer scans by matching join columns. In addition, it may take advantage of clustering on join column of the inner relation. Thus, the merging scan can remember where matching join groups are located since tuples having the same values on a join column are physically close to each other. The cost of a merge scan join can be divided into the cost of actually doing the join plus the cost of sorting the outer or inner relation if required. The cost of doing the merge is given by:

$$\begin{aligned} \text{C-merge (path1, path2)} &= \text{C-outer(path1)} \\ &+ N \times \text{C-innter(path2)} \end{aligned}$$

In case the inner relation is sorted into a temporary relation, the merging scans do not scan the entire relation looking for a match. Therefore, the cost of the inner scan can be significantly reduced comparing to nested-loop joins. The cost of the inner scan, in this case, is given by the following formula:

$$\begin{aligned} \text{C-inner(sorted list)} &= \text{TEMPPAGES}/N \\ &+ W \times \text{RSICARD} \end{aligned}$$

where TEMPPAGES is the number of pages required to hold the inner relation. RSICARD is the number of RSI calls and W is an adjustable weighting factor between I/O and CPU.

When optimizing larger queries, these join methods are used as building blocks. The System R optimizer only considers linear join trees, where join operators may occur only on one side of other join operators. Join trees with more than two relations are therefore constructed by adding a new relation to an already existing join tree. For applying the methods described above, the join tree, which is treated like a composite relation, represents the *outer* relation and the relation being added to the join tree represents the *inner* relation. The optimizer uses a dynamic programming (DP) strategy (sketched in Fig. 2) to reduce the runtime complexity for join tree construction. It organizes the optimization by the size of the join tree, initializing the DP table with single relations and then constructing larger join trees by combining smaller join trees (S_l) with new relations (R_r). For each combination of relations the best join tree found so far is stored in the DP table, which reduces the search space from $O(n!)$ to $O(n2^n)$. Further, the System R optimizer considers only join trees where a join predicate between the smaller join tree and the new relation exists (i.e., it avoids cross-products). This further reduces the search space, for example to $O(n^3)$ when the relations are simply joined in a sequence.

An important aspect of this optimization strategy is the consideration of *interesting orders*. To find solutions for joining pairs of relations, the optimizer first finds access paths for each single relation in each interesting and non-interesting tuple ordering. Recall that interesting orders are defined by GROUP BY and

```

JoinOrder ( $R = \{R_1, \dots, R_n\}$ )
for each  $R_i \in R$ 
    dpTable[ $\{R_i\}$ ] =  $R_i$ 
for each  $1 < s \leq n$  ascending // size of the join tree
    for each  $S_j \subset R$ ,  $R_r \in (R \setminus S_j) : |S_j| = s \wedge dpTable[S_j] \neq \emptyset$ 
        if  $\neg(S_j \text{ can be joined with } R_r)$  continue
         $p = dpTable[S_j] \bowtie dpTable[\{R_r\}]$ 
        if  $dpTable[S_j \cup \{R_r\}] = \emptyset \vee cost(p) < cost(dpTable[S_j \cup \{R_r\}])$ 
             $dpTable[S_j \cup \{R_r\}] = p$ 
return dpTable[ $\{R_1, \dots, R_n\}$ ]

```

System R (R^*) Optimizer. Figure 2. Dynamic programming strategy for join tree construction.

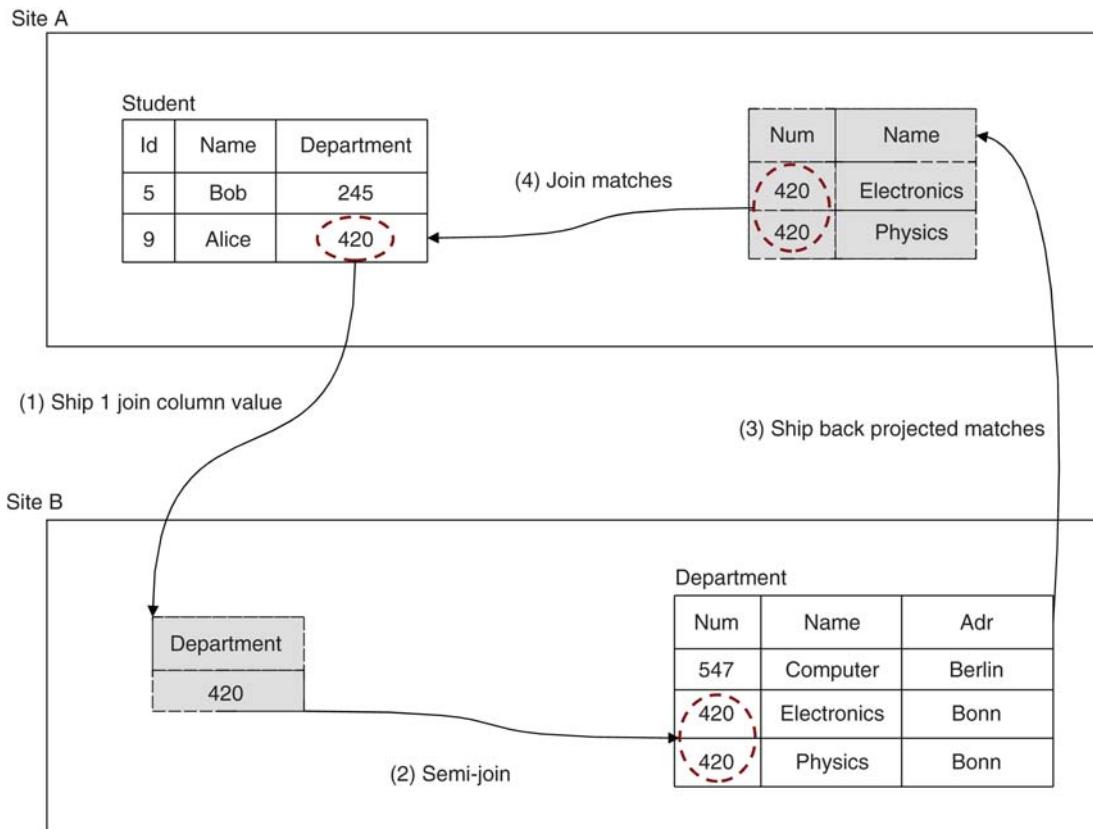
ORDER BY clauses and also by every join column. Next, the optimizer finds the best way for joining any two relations, and starts building larger join trees. For each join tree the order of the composite result is saved to allow for merge joins that would not require sorting the composite result. As the ordering can affect later operators, a plan can only be safely pruned if a cheaper one, which satisfies the same interesting ordering, is found. After join trees for all n relations have been constructed, the optimizer chooses the cheapest solution that gives the required order specified by the query. Consider an example of three relations R_1 , R_2 , R_3 in a query and the following join predicates $R_1.x = R_2.x$ and $R_2.x = R_3.x$. Assume that the costs of nested-loop and merge scan for the subquery $\{R_1, R_2\}$ are respectively C_1 and C_2 , where C_1 is lower than C_2 . Intuitively, when the optimizer looks for the best plan for $\{R_1, R_2, R_3\}$, it could consider the nested-loop method to join $\{R_1, R_2\}$ since it is the cheapest alternative. However, if the optimizer considers a merge scan to join $\{R_1, R_2\}$, the composite result will be sorted on x which may significantly reduce the cost of the join with R_3 . Thus, the optimizer has to keep track of tuple orderings that can affect the execution plans for the given query to find the optimal join tree.

R* Optimizer

The optimization algorithms described previously have been extended to efficiently process queries in a distributed database management system (R^*). In such environment, data needed by queries are stored in multiple sites. Two main factors distinguish query processing in System R from processing query in System R^* [6]. First, the communication delays, and second, the possibility of concurrent processing on multiple sites. These two factors raise the importance of developing an R^* optimizer to deal with increasing complexity of distributed query processing.

The distribution unit in R^* is a relation and each relation is stored at one site. Figure 3 shows two relations STUDENT and DEPARTMENT stored in two different sites A and B. A query is called *distributed* if it refers to relations at sites other than the query site. The simplest form of a distributed query is a query that accesses a single relation at a remote site. To execute the query, a process at the remote site accesses the relation locally and ships the query result back to the query site. In case of a join query, the R^* optimizer needs to chose a set of local and distributed parameters. The local parameters are the same as the one considered by the System R optimizer including the join method (nested-loop or merge scan), the order in which relations must be joined and the access path for each relation (index or segment scan). The distributed parameters include the choice of the join site, i.e., the site at which the join will take place, and the method for transferring a copy of the inner table to the join site, in case the inner table is not stored in the chosen join site.

R^* optimizer can use different methods to transfer tuples from a site to another one. A straightforward strategy is to ship the entire relation to the join site and store it there in a temporary table. Alternatively, the R^* optimizer can use other join methods such as semijoins, joins using hashing (Bloom) filters and joins using dynamically-created index. Semijoins, for example, help in reducing the number of transferred tuples by limiting the relevant domain. Specifically, only the tuples that could potentially match the join predicates are shipped to the join site. Figure 3 shows an example of semijoin procedure. First, it projects the outer relation to the join column and ships the results to the site of the inner relation. Second, it finds tuples from the inner relation that match the values received from the outer relation. Third, it ships a copy of the projected inner tuples to the join



System R (R*) Optimizer. Figure 3. Example of semijoin.

site. Last, it joins the received tuples to the outer relation.

Key Applications

The System R optimizer inspired many later optimizers, including the well known Starburst optimizer. Starburst uses a similar (though much more generalized) bottom-up constructive optimization technique, and eventually become the commercial database system DB2.

Cross-references

► [Query Optimization \(in Relational Databases\)](#)

Recommended Reading

1. Astrahan M.M., Blasgen M.W., Chamberlin D.D., Eswaran K.P., Gray J., Griffiths P.P., III W.F.K., Lorie R.A., McJones P.R., Mehl J.W., Putzolu G.R., Traiger I.L., Wade B.W., and Watson V. System R: Relational approach to database management. ACM Trans. Database Syst., 1(2):97–137, 1979.
2. Chamberlin D.D., Astrahan M.M., Blasgen M.W., Gray J., III W.F.K., Lindsay B.G., Lorie R.A., Mehl J.W., Price T.G., Putzolu G.R., Selinger P.G., Schkolnick M., Slutsky D.R., Traiger

I.L., Wade B.W., and Yost R.A. A history and evaluation of system R. Commun. ACM, 24(10):632–646, 1981.

3. Chamberlin D.D. and Boyce R.F. SEQUEL: A Structured English Query Language. In Proc. SIGMOD Workshop, Vol. 1. 1974, pp. 249–264.
4. Codd E.F. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387, 1970.
5. Gray J. Notes on data base operating systems. In Advanced Course: Operating Systems, 1978, pp. 393–481.
6. Lohman G.M., Mohan C., Haas L.M., Daniels D., Lindsay B.G., Selinger P.G., and Wilms P.F. Query processing in R*. In Query Processing in Database Systems, Springer, 1985, pp. 31–47.
7. Lorie R.A. and Nilsson J.F. An access specification language for a relational data base system. IBM J. Res. Dev., 23(3):286, 1979.
8. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., and Price T.G. Access path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.

System Recovery

► [Crash Recovery](#)

