A

Absolute Time

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

A temporal database contains time-referenced, or time-stamped, facts. A time reference in such a database is *absolute* if its value is independent of the context, including the current time, *now*.

Key Points

An example is "Mary's salary was raised on March 30, 2007." The fact here is that Mary's salary was raised. The absolute time reference is March 30, 2007, which is a time instant at the granularity of day.

Another example is "Mary's monthly salary was $ 15,000 from January 1, 2006 to November 30, 2007." In this example, the absolute time reference is the time period [January 1, 2006 – November 30, 2007].

Absolute time can be contrasted with *relative time*.

Cross-references

- ▶ Now in Temporal Databases
- ▶ Relative Time
- ▶ Time Instant
- ▶ Time Period
- ▶ Temporal Database
- ▶ Temporal Granularity

Recommended Reading

1. Bettini C., Dyreson C.E., Evans W.S., Snodgrass R.T., and Wang X.S. A glossary of time granularity concepts. In Temporal Databases: Research and Practice. O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS, vol. 1399. Springer, 1998, pp. 406–413.
2. 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. Sripada (eds.). LNCS, vol. 1399. Springer, 1998, pp. 367–405.

Abstract Versus Concrete Temporal Query Languages

Jan Chomicki[1], David Toman[2]
[1]State University of New York at Buffalo, Buffalo, NY, USA
[2]University of Waterloo, Waterloo, ON, Canada

Synonyms

Historical query languages

Definition

Temporal query languages are a family of query languages designed to query (and access in general) time-dependent information stored in temporal databases. The languages are commonly defined as extensions of standard query languages for non-temporal databases with *temporal features*. The additional features reflect the way dependencies of data on time are captured by and represented in the underlying temporal data model.

Historical Background

Most databases store time-varying information. On the other hand, SQL is often the language of choice for developing applications that utilize the information in these databases. Plain SQL, however, does not seem to provide adequate support for temporal applications.
Example. To represent the *employment histories* of persons, a common relational design would use a schema

`Employment(FromDate,ToDate, EID, Company)`,

with the intended meaning that a person identified by `EID` worked for `Company` continuously from `FromDate` to `ToDate`. Note that while the above schema is a standard relational schema, the additional assumption that the values of the attributes `FromDate` and `ToDate` represent *continuous periods* of time is itself *not* a part of the relational model.

Formulating even simple queries over such a schema is non-trivial. For example, the query GAPS: "*List all persons with gaps in their employment history, together*

with the gaps" leads to a rather complex formulation in, e.g., SQL over the above schema (this is left as a challenge to readers who consider themselves SQL experts; for a list of appealing, but incorrect solutions, including the reasons why, see [9]). The difficulty arises because a single tuple in the relation is conceptually a *compact representation of a set of tuples*, each tuple stating that an employment fact was true on a particular day.

The tension between the conceptual abstract temporal data model (in the example, the property that employment facts are associated with individual *time instants*) and the need for an efficient and compact representation of temporal data (in the example, the representation of continuous periods by their start and end instants) has been reflected in the development of numerous temporal data models and temporal query languages [3].

Foundations

Temporal query languages are commonly defined using *temporal extensions* of existing non-temporal query languages, such as relational calculus, relational algebra, or SQL. The temporal extensions can be categorized in two, mostly orthogonal, ways:

- *The choice of the actual temporal values manipulated by the language*. This choice is primarily determined by the underlying temporal data model. The model also determines the associated operations on these values. The meaning of temporal queries is then defined in terms of temporal values and operations on them, and their interactions with *data* (non-temporal) values in a temporal database.
- *The choice of syntactic constructs to manipulate temporal values in the language*. This distinction determines whether the temporal values in the language are accessed and manipulated *explicitly*, in a way similar to other values stored in the database, or whether the access is *implicit*, based primarily on *temporally extending* the meaning of constructs that already exist in the underlying non-temporal language (while still using the operations defined by the temporal data model).

Additional design considerations relate to *compatibility* with existing query languages, e.g., the notion of temporal upward compatibility.

However, as illustrated above, an additional hurdle stems from the fact that many (early) temporal query languages allowed the users to manipulate a *finite underlying representation* of temporal databases rather than the actual temporal values/objects in the associated temporal data model. A typical example of this situation would be an approach in which the temporal data model is based on time instants, while the query language introduces interval-valued attributes. Such a discrepancy often leads to a complex and unintuitive semantics of queries.

In order to clarify this issue, Chomicki has introduced the notions of *abstract* and *concrete* temporal databases and query languages [2]. Intuitively, *abstract temporal query languages* are defined at the conceptual level of the temporal data model, while their *concrete* counterparts operate directly on an actual *compact encoding* of temporal databases. The relationship between abstract and concrete temporal query languages is also implicitly present in the notion of snapshot equivalence [7]. Moreover, Bettini et al. [1] proposed to distinguish between *explicit* and *implicit* information in a temporal database. The explicit information is stored in the database and used to derive the implicit information through *semantic assumptions*. Semantic assumptions related to fact persistence play a role similar to mappings between concrete and abstract databases, while other assumptions are used to address time-granularity issues.

Abstract Temporal Query Languages

Most temporal query languages derived by temporally extending the relational calculus can be classified as abstract temporal query languages. Their semantics are defined in terms of abstract temporal databases which, in turn, are typically defined within the point-stamped temporal data model, in particular *without* any additional hidden assumptions about the meaning of tuples in instances of temporal relations.

Example. The *employment histories* in an abstract temporal data model would most likely be captured by a simpler schema "`Employment(Date, EID, Company)`", with the intended meaning that a person identified by `EID` was working for `Company` on a particular `Date`. While instances of such a schema can potentially be very large (especially when a fine granularity of time is used), formulating queries is now much more natural.

Choosing abstract temporal query languages over concrete ones resolves the first design issue: the temporal values used by the former languages are time instants equipped with an appropriate temporal ordering (which

is typically a linear order over the instants), and possibly other predicates such as temporal distance. The second design issue – access to temporal values – may be resolved in two different ways, as exemplified by two different query languages. They are as follows:

- Temporal Relational Calculus (TRC): a two-sorted first-order logic with variables and quantifiers explicitly ranging over the time and data domains.
- First-order Temporal Logic (FOTL): a language with an implicit access to timestamps using temporal connectives.

Example. The GAPS query is formulated as follows:

TRC: $\exists t_1,t_3.t_1 < t_2 < t_3 \wedge \exists c.\texttt{Employment}(t_1, x, c) \wedge$
$(\neg \exists c.\texttt{Employment}(t_2, x, c)) \wedge$
$\exists c.\texttt{Employment}(t_3, x, c);$

FOTL: $\blacklozenge \exists c.\texttt{Employment}(x,c) \wedge$
$(\neg \exists c.\texttt{Employment}(x, c)) \wedge$
$\Diamond \exists c.\texttt{Employment}(x, c)$

Here, the explicit access to temporal values (in TRC) using the variables t_1, t_2, and t_3 can be contrasted with the implicit access (in FOTL) using the temporal operators $\blacklozenge$ (read "sometime in the past") and $\Diamond$ (read "sometime in the future"). The conjunction in the FOTL query represents an implicit temporal join. The formulation in TRC leads immediately to an equivalent way of expressing the query in SQL/TP [9], an extension of SQL based on TRC.

Example. The above query can be formulated in SQL/TP as follows:

```
SELECT t.Date, e1.EID
FROM Employment e1, Time t, Employment e2
WHERE e1.EID = e2.EID
AND e1.Date < e2.Date
  AND NOT EXISTS ( SELECT *
    FROM Employment e3
    WHERE e1.EID = e3.EID
    AND t.Date = e3.Date
      AND e1.Date < e3.Date
      AND e3.Date < e2.Date )
```

The unary constant relation `Time` contains all time instants in the time domain (in our case, all `Dates`) and is only needed to fulfill syntactic SQL-style requirements on attribute ranges. However, despite the fact that the instance of this relation is not finite, the query can be efficiently evaluated [9].

Note also that in all of the above cases, the formulation is *exactly the same* as if the underlying temporal database used the *plain* relational model (allowing for attributes ranging over time instants).

The two languages, FOTL and TRC, are the counterparts of the snapshot and timestamp models (cf. the entry Point-stamped Data Models) and are the roots of many other temporal query languages, ranging from the more TRC-like temporal extensions of SQL to more FOTL-like temporal relational algebras (e.g., the conjunction in temporal logic directly corresponds to a temporal join in a temporal relational algebra, as both of them induce an *implicit equality* on the associated time attributes).

Temporal integrity constraints over point-stamped temporal databases can also be conveniently expressed in TRC or FOTL.

Multiple Temporal Dimensions and Complex Values.

While the abstract temporal query languages are typically defined in terms of the point-based temporal data model, they can similarly be defined with respect to complex temporal values, e.g., pairs (or tuples) of time instants or even sets of time instants. In these cases, particularly in the case of set-valued attributes, it is important to remember that the set values are treated as *indivisible objects*, and hence truth (i.e., query semantics) is associated with the entire objects, but not necessarily with their components/subparts.

Concrete Temporal Query Languages

Although abstract temporal query languages provide a convenient and clean way of specifying queries, they are not immediately amenable to implementation. The main problem is that, in practice, the facts in temporal databases persist over periods of time. Storing all true facts individually *for every time instant* during a period would be prohibitively expensive or, in the case of infinite time domains such as *dense time*, even impossible.

Concrete temporal query languages avoid these problems by operating directly on the compact encodings of temporal databases. The most commonly used encoding is the one that uses *intervals*. However, in this setting, a tuple that associates a fact with such an interval is a compact representation of the association between the same fact and *all the time instants that belong to this interval*. This observation leads to the design choices that are commonly present in such languages:

- Coalescing is used, explicitly or implicitly, to consolidate representations of (sets of) time instants associated *with the same fact*. In the case of interval-based encodings, this leads to coalescing adjoining or overlapping intervals into a single interval. Note that coalescing only changes the *concrete representation* of a temporal relation, not its meaning (i.e., the abstract temporal relation); hence it has no counterpart in abstract temporal query languages.
- Implicit *set operations* on time values are used in relational operations. For example, conjunction (join) typically uses set intersection to generate a compact representation of the time instants attached to the facts in the result of such an operation.

Example. For the running example, a concrete schema for the employment histories would typically be defined as "`Employment(VT, EID, Company)`" where `VT` is a valid time attribute ranging over periods (intervals). The GAPS query can be formulated in a calculus-style language corresponding to TSQL2 (see the entry on TSQL2) along the following lines:

$$\exists I_1, I_2.[\exists c.\texttt{Employment}(I_1, x, c)] \wedge [\exists c.\texttt{Employment}(I_2, x, c)] \wedge I_1 \texttt{ precedes } I_2 \wedge I = [\texttt{end}(I_1) + 1, \texttt{begin}(I_2) - 1].$$

In particular, the variables I_1 and I_2 range over periods and the `precedes` relationship is one of Allen's interval relationships. The final conjunct,

$$I = [\texttt{end}(I_1) + 1, \texttt{begin}(I_2) - 1],$$

creates a new period corresponding to the time instants related to a person's *gap in employment*; this interval value is explicitly constructed from the end and start points of I_1 and I_2, respectively. For the query to be correct, however, the results of evaluating the bracketed subexpressions, e.g., "$[\exists c.\texttt{Employment}(I_1, x, c)]$," have to be *coalesced*. Without the insertion of the explicit coalescing operators, the query is *incorrect*. To see that, consider a situation in which a person p_0 is first employed by a company c_1, then by c_2, and finally by c_3, without any gaps in employment. Then without coalescing of the bracketed subexpressions of the above query, p_0 will be returned as a part of the result of the query, which is incorrect. Note also that it is not enough for the underlying (concrete) database to be coalesced.

The need for an explicit use of coalescing often makes the formulation of queries in some concrete SQL-based temporal query languages cumbersome and error-prone.

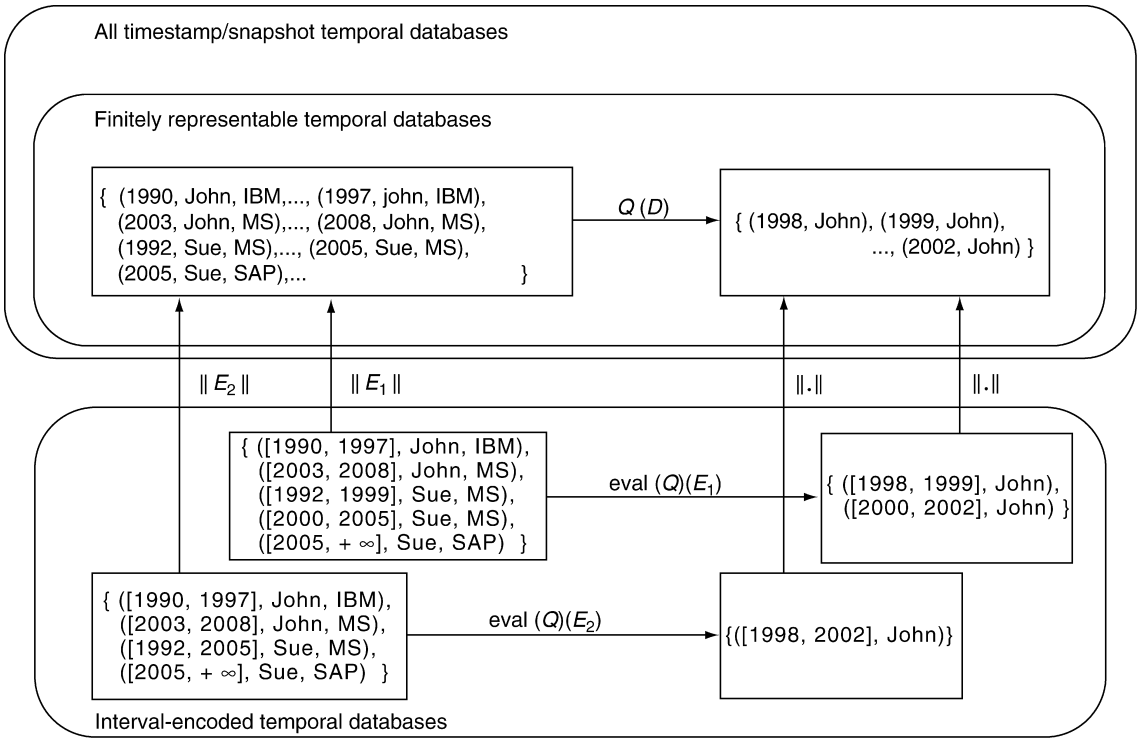
An orthogonal issue is the difference between explicit and implicit access to temporal values. This distinction also carries over to the concrete temporal languages. Typically, the various temporal extensions of SQL are based on the assumption of an explicit access to temporal values (often employing a built-in *valid time* attribute ranging over intervals or temporal elements), while many temporal relational algebras have chosen to use the implicit access based on temporally extending standard relational operators such as temporal join or temporal projection.

Compilation and Query Evaluation. An alternative to allowing users direct access to the encodings of temporal databases is to develop techniques that allow the evaluation of *abstract temporal queries* over these encodings. The main approaches are based on *query compilation* techniques that map abstract queries to concrete queries, while preserving query answers. More formally:

$$Q(\|E\|) = \|\texttt{eval}(Q)(E)\|,$$

where Q an abstract query, eval(Q) the corresponding concrete query, E is a concrete temporal database, and $\|.\|$ a mapping that associates encodings (concrete temporal databases) with their abstract counterparts (cf. Fig. 1). Note that a single abstract temporal database, D, can be encoded using several *different* instances of the corresponding concrete database, e.g., E_1 and E_2 in Fig. 1.

Most of the practical temporal data models adopt a common approach to physical representation of temporal databases: with every fact (usually represented as a tuple), a *concise encoding* of the set of time points at which the fact holds is associated. The encoding is commonly realized by *intervals* [6,7] or temporal elements (finite unions of intervals). For such an encoding it has been shown that both First-Order Temporal Logic [4] and Temporal Relational Calculus [8] queries can be *compiled* to first-order queries over a natural relational representation of the interval encoding of the database. Evaluating the resulting queries yields the interval encodings of the answers to the original queries, as if the queries were directly evaluated on the point-stamped temporal database. Similar results can be obtained for more complex encodings, e.g., periodic



Abstract Versus Concrete Temporal Query Languages. Figure 1. Query evaluation over interval encodings of point-stamped temporal databases.

sets, and for abstract temporal query languages that adopt the duplicate semantics matching the SQL standard, such as SQL/TP [9].

Key Applications

Temporal query languages are primarily used for querying temporal databases. However, because of their generality they can be applied in other contexts as well, e.g., as an underlying conceptual foundation for querying sequences and data streams [5].

Cross-references

- ▶ Allen's Relations
- ▶ Bitemporal Relation
- ▶ Constraint Databases
- ▶ Key
- ▶ Nested Transaction Models
- ▶ Non First Normal Form
- ▶ Point-Stamped Temporal Models
- ▶ Relational Model
- ▶ Snapshot Equivalence
- ▶ SQL
- ▶ Telic Distinction in Temporal Databases
- ▶ Temporal Coalescing
- ▶ Temporal Data Models
- ▶ Temporal Element
- ▶ Temporal Granularity
- ▶ Temporal Integrity Constraints
- ▶ Temporal Joins
- ▶ Temporal Logic in Database Query Languages
- ▶ Temporal Relational Calculus
- ▶ Time Domain
- ▶ Time Instant
- ▶ Transaction Time
- ▶ TSQL2
- ▶ Valid Time

Recommended Reading

1. Bettini C., Wang X.S., and Jajodia S. Temporal Semantic Assumptions and Their Use in Databases. Knowl. Data Eng., 10(2):277–296, 1998.
2. Chomicki J. Temporal query languages: a survey. In Proc. 1st Int. Conf. on Temporal Logic, 1994, pp. 506–534.
3. Chomicki J. and Toman D. Temporal databases. In Handbook of Temporal Reasoning in Artificial Intelligence, Fischer M., Gabbay D., and Villa L. (eds.). Elsevier Foundations of Artificial Intelligence, 2005, pp. 429–467.

4. Chomicki J., Toman D., and Böhlen M.H. Querying ATSQL databases with temporal logic. ACM Trans. Database Syst., 26(2):145–178, 2001.
5. Law Y.-N., Wang H., and Zaniolo C. Query languages and data models for database sequences and data streams. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 492–503.
6. Navathe S.B. and Ahmed R. In Temporal Extensions to the Relational Model and SQL. Tansel A., Clifford J., Gadia S., Jajodia S., Segev A., and Snodgrass R.T. (eds.). Temporal Databases: Theory, Design, and Implementation. Benjamin/Cummings, Menlo Park, CA, 1993, pp. 92–109.
7. Snodgrass R.T. The temporal query language TQuel. ACM Trans. Database Syst., 12(2):247–298, 1987.
8. Toman D. Point vs. interval-based query languages for temporal databases. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1996, pp. 58–67.
9. Toman D. Point-based temporal extensions of SQL. In Proc. 5th Int. Conf. on Deductive and Object-Oriented Databases, 1997, pp. 103–121.

Abstraction

Bernhard Thalheim
Christian-Albrechts University Kiel, Kiel, Germany

Synonyms

Component abstraction; Implementation abstraction; Association; Aggregation; Composition; Grouping; Specialization; Generalisation; Classification

Definition

Abstraction allows developers to concentrate on the essential, relevant, or important parts of an application. It uses a mapping to a model from things in reality or from virtual things. The model has the truncation property, i.e., it lacks some of the details in the original, and a pragmatic property, i.e., the model use is only justified for particular model users, tools of investigation, and periods of time. Database engineering uses construction abstraction, context abstraction, and refinement abstraction. Construction abstraction is based on the principles of hierarchical structuring, constructor composition, and generalization. Context abstraction assumes that the surroundings of a concept are commonly understood by a community or within a culture and focuses on the concept, turning away attention from its surroundings such as the environment and setting. Refinement abstraction uses the principle of modularization and information hiding. Developers typically use conceptual models or languages for representing and conceptualizing abstractions. The enhanced entity-relationship model schema are typically depicted by an EER diagram.

Key Points

Database engineering distinguishes three kinds of abstraction: construction abstraction, context abstraction, and refinement abstraction.

Constructor composition depends on the constructors as originally introduced by J. M. Smith and D.C.W. Smith. Composition constructors must be well founded and their semantics must be derivable by inductive construction. There are three main methods for construction: development of ordered structures on the basis of hierarchies, construction by combination or association, and construction by classification into groups or collections. The set constructors $\subset$ (subset), $\times$ (product), and $\mathcal{P}$ (powerset) for subset, product and nesting are complete for the construction of sets.

Subset constructors support hierarchies of object sets in which one set of objects is a subset of some other set of objects. Subset hierarchies are usually a rooted tree. Product constructors support associations between object sets. The schema is decomposed into object sets related to each other by association or relationship types. Power set constructors support a classification of object sets into clusters or groups of sets – typically according to their properties.

Context abstraction allows developers to commonly concentrate on those parts of an application that are essential for some perspectives during development and deployment of systems. Typical types of context abstraction are component abstraction, separation of concern, interaction abstraction, summarization, scoping, and focusing on typical application cases.

Component abstraction factors out repeating, shared or local patterns of components or functions from individual concepts. It allows developers to concentrate on structural or behavioral aspects of similar elements of components. Separation of concern allows developers to concentrate on those concepts under development and to neglect all other concepts that are stable or not under consideration. Interaction abstraction allows developers to concentrate on parts of the model that are essential for interaction with other systems or users. Summarisation maps the conceptualizations within the scope to more abstract concepts. Scoping is typically used to select those concepts that are necessary for current development and removes

those concepts which that do not have an impact on the necessary concepts.

Database models may cover a large variety of different application cases. Some of them reflect exceptional, abnormal, infrequent and untypical application situations. Focusing on typical application cases explicitly separates models intended for the normal or typical application case from those that are atypical. Atypical application cases are not neglected but can be folded into the model whenever atypical situations are considered.

The context abstraction concept is the main concept behind federated databases. Context of databases can be characterized by schemata, version, time, and security requirements. Sub-schemata, types of the schemata or views on the schemata, are associated with explicit import/export bindings based on a name space. Parametrization lets developers consider collections of objects. Objects are identifiable under certain assumptions and completely identifiable after instantiation of all parameters.

Interaction abstraction allows developers to display the same set of objects in different forms. The view concept supports this visibility concept. Data is abstracted and displayed in various levels of granularity. Summarization abstraction allows developers to abstract from details that are irrelevant at a certain point. Scope abstraction allows developers to concentrate on a number of aspects. Names or aliases can be multiply used with varying structure, functionality and semantics.

Refinement abstraction mainly concerns implementation and modularisation. It allows developers to selectively retain information about structures. Refinement abstraction is defined on the basis of the development cycle (refinement of implementations). It refines, summarizes and views conceptualizations, hides or encapsulates details, or manages collections of versions. Each refinement step transforms a schema to a schema of finer granularity. Refinement abstraction may be modeled by refinement theory and infomorphisms.

Encapsulation removes internal aspects and concentrates on interface components. Blackbox or graybox approaches hide all aspects of the objects being considered. Partial visibility may be supported by modularization concepts. Hiding supports differentiation of concepts into public, private (with the possibility to be visible as "friends") and protected (with visibility to subconcepts). It is possible to define a number of visibility conceptualizations based on inflection. Inflection is used for the injection of combinable views into the given view, for tailoring, ordering and restructuring of views, and for enhancement of views by database functionality. Behavioral transparency is supported by the glassbox approach. Security views are based on hiding. Versioning allows developers to manage a number of concepts which can be considered to be versions of each other.

Cross-references

- ▶ Entity Relationship Model
- ▶ Extended Entity-Relationship Model
- ▶ Language Models
- ▶ Object Data Models
- ▶ Object-Role Modeling
- ▶ Specialization and Generalization

Recommended Reading

1. Börger E. The ASM refinement method. Formal Aspect. Comput., 15:237–257, 2003.
2. Smith J.M. and Smith D.C.W. Data base abstractions: aggregation and generalization. ACM Trans. Database Syst., 2 (2):105–133, 1977.
3. Thalheim B. Entity-Relationship Modeling – Foundations of Database Technology. Springer, 2000.

Access Control

Elena Ferrari
University of Insubria, Varese, Italy

Synonyms

Authorization verification

Definition

Access control deals with preventing unauthorized operations on the managed data. Access control is usually performed against a set of *authorizations* stated by Security Administrators (SAs) or users according to the *access control policies* of the organization. Authorizations are then processed by the *access control mechanism* (or *reference monitor*) to decide whether each access request can be authorized or should be denied.

Historical Background

Access control models for DBMSs have been greatly influenced by the models developed for the protection of operating system resources. For instance, the model

proposed by Lampson [16] is also known as the *access matrix* model since authorizations are represented as a matrix. However, much of the early work on database protection was on inference control in statistical databases.

Then, in the 1970s, as research in relational databases began, attention was directed towards access control issues. As part of the research on System R at IBM Almaden Research Center, there was much work on access control for relational database systems [11,15], which strongly influenced access control models and mechanisms of current commercial relational DBMSs. Around the same time, some early work on multilevel secure database management systems (MLS/DBMSs) was reported. However, it was only after the Air Force Summer Study in 1982 [1] that developments on MLS/DBMSs began. For instance, the early prototypes based on the integrity lock mechanisms developed at the MITRE Corporation. Later, in the mid-1980s, pioneering research was carried out at SRI International and Honeywell Inc. on systems such as SeaView and LOCK Data Views [9]. Some of the technologies developed by these research efforts were transferred to commercial products by corporations such as Oracle, Sybase, and Informix. In the 1990s, numerous other developments were made to meet the access control requirements of new applications and environments, such as the World Wide Web, data warehouses, data mining systems, multimedia systems, sensor systems, workflow management systems, and collaborative systems. This resulted in several extensions to the basic access control models previously developed, by including the support for temporal constraints, derivation rules, positive and negative authorizations, strong and weak authorizations, and content and context-dependent authorizations [14]. Role-based access control has been proposed [12] to simplify authorization management within companies and organizations. Recently, there have been numerous developments in access control, mainly driven by developments in web data management. For example, standards such as XML (eXtensible Markup Language) and RDF (Resource Description Framework) require proper access control mechanisms [7]. Also, web services and the semantic web are becoming extremely popular and therefore research is currently carried out to address the related access control issues [13]. Access control is currently being examined for new application areas, such as knowledge management [4], data outsourcing, GIS [10], peer-to-peer computing and stream data management [8]. For example, in the case of knowledge management applications, it is important to protect the intellectual property of an organization, whereas when data are outsourced, it is necessary to allow the owner to enforce its access control policies, even if data are managed by a third party.

Foundations

The basic building block on which access control relies is a set of *authorizations*: which state, who can access which resource, and under which mode. Authorizations are specified according to a set of *access control policies*, which define the high-level rules according to which access control must occur. In its basic form, an authorization is, in general, specified on the basis of three components (s,o,p), and specifies that subject s is authorized to exercise privilege p on object o. The three main components of an authorization have the following meaning:

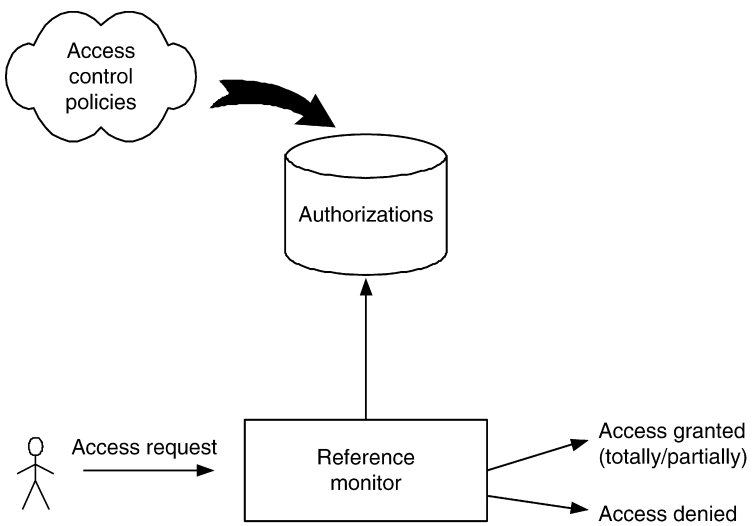
- *Authorization subjects*: They are the "active" entities in the system to which authorizations are granted. Subjects can be further classified into the following, not mutually exclusive, categories: *users*, that is, single individuals connecting to the system; *groups*, that is, sets of users; roles, that is, named collection of privileges needed to perform specific activities within the system; and *processes*, executing programs on behalf of users.
- *Authorization objects*: They are the "passive" components (i.e., resources) of the system to which protection from unauthorized accesses should be given. The set of objects to be protected clearly depends on the considered environment. For instance, files and directories are examples of objects of an operating system environment, whereas in a relational DBMS, examples of resources to be protected are relations, views and attributes. Authorizations can be specified at different granularity levels, that is, on a whole object or only on some of its components. This is a useful feature when an object (e.g., a relation) contains information (e.g., tuples) of different sensitivity levels and therefore requires a differentiated protection.
- *Authorization privileges*: They state the types of operations (or access modes) that a subject can exercise on the objects in the system. As for objects, the set of privileges also depends on the resources

to be protected. For instance, read, write, and execute privileges are typical of an operating system environment, whereas in a relational DBMS privileges refer to SQL commands (e.g., select, insert, update, delete). Moreover, new environments such as digital libraries are characterized by new access modes, for instance, usage or copying access rights.

Depending on the considered domain and the way in which access control is enforced, objects, subjects and/or privileges can be hierarchically organized. The hierarchy can be exploited to propagate authorizations and therefore to simplify authorization management by limiting the set of authorizations that must be explicitly specified. For instance, when objects are hierarchically organized, the hierarchy usually represents a "part-of" relation, that is, the hierarchy reflects the way objects are organized in terms of other objects. In contrast, the privilege hierarchy usually represents a subsumption relation among privileges. Privileges towards the bottom of the hierarchy are subsumed by privileges towards the top (for instance, the write privilege is at a higher level in the hierarchy with respect to the read privilege, since write subsumes read operations). Also roles and groups can be hierarchically organized. The group hierarchy usually reflects the membership of a group to another group. In contrast, the role hierarchy usually reflects the relative position of roles within an organization. The higher the level of a role in the hierarchy, the higher its position in the organization.

Authorizations are stored into the system and are then used to verify whether an access request can be authorized or not. How to represent and store authorizations depends on the protected resources. For instance, in a relational DBMS, authorizations are modeled as tuples stored into system catalogs. In contrast, when resources to be protected are XML documents, authorizations are usually encoded using XML itself. Finally, the last key component of the access control infrastructure is the *access control mechanism* (or *reference monitor*), which is a trusted software module in charge of enforcing access control. It intercepts each access request submitted to the system (for instance, SQL statements in case of relational DBMSs) and, on the basis of the specified authorizations, it determines whether the access can be partially or totally authorized or should be denied. The reference monitor should be *non-bypassable*. Additionally, the hardware and software architecture should ensure that the reference monitor is *tamper proof*, that is, it cannot be maliciously modified (or at least that any improper modification can be detected). The main components of access control are illustrated in Fig. 1.

A basic distinction when dealing with access control is between *discretionary* and *mandatory* access control. Discretionary access control (DAC) governs the access of subjects to objects on the basis of subjects' identity and a set of explicitly specified authorizations that specify, for each subject, the set of objects that



Access Control. Figure 1. Access control: main components.

he/she can access in the system and the allowed access modes. When an access request is submitted to the system, the access control mechanism verifies whether or not the access can be authorized according to the specified authorizations. The system is discretionary in the sense that a subject, by proper configuring the set of authorizations, is both able to enforce various access control requirements and to dynamically change them when needed (simply by updating the authorization state). In contrast, mandatory access control (MAC) specifies the accesses that subjects can exercise on the objects in the system, on the basis of subjects and objects security classification [14]. Security classes usually form a partially ordered set. This type of security has also been referred to as *multilevel security*, and database systems that enforce multilevel access control are called *Multilevel Secure Database Management Systems* (MLS/DBMSs). When mandatory access control is enforced, authorizations are implicitly specified, by assigning subjects and objects proper security classes. The decision on whether or not to grant an access depends on the access mode and the relation existing between the classification of the subject requesting the access and that of the requested object. In addition to DAC and MAC, role-based access control (RBAC) has been more recently proposed [12]. RBAC is an alternative to discretionary and mandatory access control, mainly conceived for regulating accesses within companies and organizations. In RBAC, permissions are associated with roles, instead of with users, and users acquire permissions through their membership to roles. The set of authorizations can be inferred by the sets of user-role and role-permission assignments.

Key Applications

Access control techniques are applied in almost all environments that need to grant a controlled access to their resources, including, but not limited, to the following: DBMSs, Data Stream Management Systems, Operat ing Systems, Workflow Management Systems, Digital Libraries, GIS, Multimedia DBMSs, E-commerce services, Publish-subscribe systems, Data warehouses.

Future Directions

Altough access control is a mature area with consolidated results, the evolution of DBMSs and the requirements of new applications and environments pose new challenges to the research community. An interesting discussion on open research issues in the field can be found in [6]. Some research issues which complement those presented in [6] are discussed below.

Social networks. Web-based social networks (WBSNs) are online communities where participants can establish relationships and share resources across the web with other users. In recent years, several WBSNs have been adopting semantic web technologies, such as FOAF, for representing users' data and relationships, making it possible to enforce information interchange across multiple WBSNs. Despite its advantages in terms of information diffusion, this raised the need for giving content owners more control on the distribution of their resources, which may be accessed by a community far wider than they expected. So far, this issue has been mainly addressed in a very simple way, by some of the available WBSNs, by only allowing users to state whether a specific information (e.g., personal data and resources) should be public or accessible only by the users with whom the owner of such information has a direct relationship. Such simple access control strategies have the advantage of being straightforward, but they are not flexible enough in denoting authorized users. In fact, they do not take into account the type of the relationships existing between users and, consequently, it is not possible to state that only, say, my "friends" can access a given information. Moreover, they do not allow to grant access to users who have an indirect relationship with the resource owner (e.g., the "friends of my friends"). Therefore, more flexible mechanisms are needed, making a user able to decide which network participants are authorized to access his/her resources and personal information. Additionally, since the number of social network users is considerably higher than those in conventional DBMSs, the traditional server-side way of enforcing access control, that is, the one relying on a centralized trusted reference monitor, should be revised and more efficient and distributed strategies should be devised for WBSNs. Until now, apart from [3], most of the security research on WBSNs has focused on privacy-preserving mining of social network data. The definition of a comprehensive framework for efficiently enforcing access control in social networks is therefore still an issue to be investigated.

- *Data streams.* In many applications, such as telecommunication, battle field monitoring, network

monitoring, financial monitoring, sensor networks, data arrive in the form of high speed data streams. These data typically contain sensitive information (e.g., health information, credit card numbers) and thus unauthorized accesses should be avoided. Although many data stream processing systems have been developed so far (e.g., Aurora, Borealis, STREAM, TelegraphCQ, and StreamBase), the focus of these systems has been mainly on performance issues rather than on access control. On the other hand, though the data security community has a very rich history in developing access control models [9], these models are largely tailored to traditional DBMSs and therefore they cannot be readily applied to data stream management systems [8]. This is mainly because: (i) traditional data are static and bounded, while data streams are unbounded and infinite; (ii) queries in traditional DBMSs are one time and ad-hoc, whereas queries over data streams are typically continuous and long running; (iii) in traditional DBMSs, access control is enforced when users access the data; (iv) in data stream applications access control enforcement is data-driven (i.e., whenever data arrive), as such access control is more computational intensive in data stream applications and specific techniques to handle it efficiently should be devised; (v) temporal constraints (e.g., sliding windows) are more critical in data stream applications than in traditional DBMSs.

- *Semantic web.* The web is now evolving into the semantic web. The semantic web [5] is a web that is intelligent with machine-readable web pages. The major components of the semantic web include web infrastructures, web databases and services, ontology management and information integration. There has been much work on each of these areas. However, very little work has been devoted to access control. If the semantic web is to be effective, it is necessary to ensure that the information on the web is protected from unauthorized accesses and malicious modifications. Also, it must be ensured that individual's privacy is maintained. To cope with these issues, it is necessary to secure all the semantic web related technologies, such as XML, RDF, Agents, Databases, web services, and Ontologies and ensure the secure interoperation of all these technologies [13].

Cross-references

- ▶ Access Control Policy Languages
- ▶ Discretionary Access Control
- ▶ Mandatory Access Control
- ▶ Multilevel Secure Database Management System
- ▶ Role Based Access Control
- ▶ Storage Security

Recommended Reading

1. Air Force Studies Board, Committee on Multilevel Data Management Security. Multilevel data management security. National Research Council, 1983.
2. Berners-Lee T. et al. The semantic web. Scientific American, 2001.
3. Bertino E., and Sandhu R.S. Database security: concepts, approaches, and challenges. IEEE Trans. Dependable and Secure Computing, 2(1):2–19, 2005.
4. Bertino E., Khan L.R., Sandhu R.S., and Thuraisingham B.M. Secure knowledge management: confidentiality, trust, and privacy. IEEE Trans. Syst. Man Cybern. A, 36(3):429–438, 2006.
5. Carminati B., Ferrari E., and Perego A. Enforcing access control in web-based social networks. ACM trans. Inf. Syst. Secur., to appear.
6. Carminati B., Ferrari E., and Tan K.L. A framework to enforce access control over Data Streams. ACM Trans. Inf. Syst. Secur., to appear.
7. Carminati B., Ferrari E., and Thuraisingham B.M. Access control for web data: models and policy languages. Ann. Telecomm., 61 (3–4):245–266, 2006.
8. Carminati B., Ferrari E., and Bertino E. Securing XML data in third party distribution systems. In Proc. of the ACM Fourteenth Conference on Information and Knowledge Management, 2005.
9. Castano S., Fugini M.G., Martella G., and Samarati P. Database security. Addison Wesley, 1995.
10. Damiani M.L. and Bertino E. Access control systems for geo-spatial data and applications. In Modelling and management of geographical data over distributed architectures, A. Belussi, B. Catania, E. Clementini, E. Ferrari (eds.). Springer, 2007.
11. Fagin R. On an authorization mechanism. ACM Trans. Database Syst., 3(3):310–319, 1978.
12. Ferraiolo D.F., Sandhu R.S., Gavrila S.I., Kuhn D.R., and Chandramouli R. Proposed NIST standard for role-based access control. ACM Trans. Inf. Syst. Secur., 4(3):224–274, 2001.
13. Ferrari E. and Thuraisingham B.M. Security and privacy for web databases and services. In Advances in Database Technology, Proc. 9th Int. Conf. on Extending Database Technology, 2004, pp. 17–28.
14. Ferrari E. and Thuraisingham B.M. Secure database systems. In O. Diaz, M. Piattini (eds.). Advanced databases: technology and design. Artech House, 2000.
15. Griffiths P.P. and Wade B.W. An authorization mechanism for a relational database system. ACM Trans. Database Syst., 1 (3):242–255, 1976.
16. Lampson B.W. Protection. Fifth Princeton Symposium on Information Science and Systems, Reprinted in ACM Oper. Sys. Rev., 8(1):18–24, 1974.

Access Control Administration Policies

Elena Ferrari
University of Insubria, Varese, Italy

Synonyms

Authorization administration policies; Authorization administration privileges

Definition

Administration policies regulate who can modify the authorization state, that is, who has the right to grant and revoke authorizations.

Historical Background

Authorization management is a an important issue when dealing with access control and, as such, research on this topic is strongly related to the developments in access control. A milestone in the field is represented by the research carried out in the 1970s at IBM in the framework of the System R project. In particular, the work by Griffiths and Wade [9] defines a semantics for authorization revocation, which had greatly influenced the way in which authorization revocation has been implemented in commercial Relational DBMSs. Administrative policies for Object-oriented DBMSs have been studied in [8]. Later on, some extensions to the System R access control administration model, have been defined [3], with the aim of making it more flexible and adaptable to a variety of access control requirements. Additionally, as the research on extending the System R access control model with enhanced functionalities progresses, authorization administration has been studied for these extensions, such as temporal authorizations [2], strong and weak and positive and negative authorizations [4]. Also, administrative policies for new environments and data models such as WFMSs [1] and XML data [12] have been investigated. Back in the 1990s, when research on role-based access control began, administration policies for RBAC were investigated [6,11,10,13]. Some of the ideas developed as part of this research were adopted by the current SQL:2003 standard [7].

Foundations

Access control administration deals with granting and revoking of authorizations. This function is usually regulated by proper *administration policies*. Usually, if mandatory access control is enforced, the adopted administration policies are very simple, so that the Security Administrator (SA) is the only one authorized to change the classification level of subjects and objects. In contrast, discretionary and role-based access control are characterized by more articulated administration policies, which can be classified according to the following categories [3]:

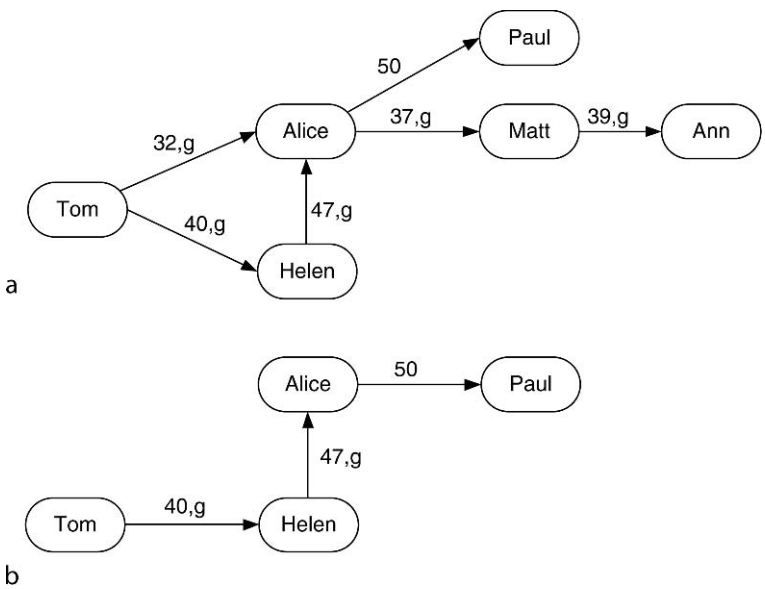
- *SA administration*. According to this policy, only the SA can grant and revoke authorizations. Although the SA administration policy has the advantage of being very simple and easily implemented, it has the disadvantage of being highly centralized (even though different SAs can manage different portions of the database) and is seldom used in current DBMSs, apart from very simple systems.
- *Object owner administration*. This is the policy commonly adopted by DBMSs and operating systems. Under this policy, whoever creates an object become its owner and he/she is the only one authorized to grant and revoke authorizations on the object.
- *Joint administration*. Under this policy, particularly suited for collaborative environments, several subjects are jointly responsible for administering specific authorizations. For instance, under the joint administration policy it can be a requirement that the authorization to write a certain document is given by two different users, such as two different job functions within an organization. Authorizations for a subject to access a data object requires that all the administrators of the object issue a grant request.

The object owner administration policy can be further combined with *administration delegation*, according to which the administrator of an object can grant other subjects the right to grant and revoke authorizations on the object. Delegation can be specified for selected privileges, for example only for read operations. Most current DBMSs support the owner administration policy with delegation. For instance, the `Grant` command provided by the SQL:2003 standard [7] supports a `Grant Option` optional clause. If a privilege p is granted with the grant option on an object o, the subject receiving it is not only authorized to exercise p on object o but he/she is also authorized to grant other subjects authorizations for p on object o with or without the grant option. Moreover, SQL:2003 provides an optional `Admin Option` clause, which has

the same meaning as the `Grant option` clause but it applies to roles instead of to standard authorizations. If a subject is granted the authorization to play a role with the admin option he/she not only receives all the authorizations associated with the role, but he/she can also authorize other subjects to play that role.

If administration delegation is supported, different administrators can grant the same authorization to the same subject. A subject can therefore receive an authorization for the same privilege on the same object by different sources. An important issue is therefore related to the management of revoke operations, that is, what happens when a subject revokes some of the authorizations he/she previously granted. For instance, consider three users: `Ann`, `Tom`, and `Alice`. Suppose that `Ann` grants `Tom` the privilege to select tuples from the `Employee` relation with the grant option and that, by having this authorization, `Tom` grants `Alice` the same privilege on the `Employee` relation. What happens to the authorization of `Alice` when `Ann` revokes `Tom` the privilege to select tuples from the `Employee` relation? The System R authorization model [9] adopts the most conscious approach with respect to security by enforcing *recursive revocation*: whenever a subject revokes an authorization on a relation from another subject, all the authorizations that the revokee had granted because of the revoked authorization are recursively removed from the system. The revocation is iteratively applied to all the subjects that received an authorization from the revokee. In the example above, `Alice` will lose the privilege to select tuples from the `Employee` relation when `Ann` revokes this privilege to `Tom`.

Implementing recursive revocation requires keeping track of the *grantor* of each authorization, that is, the subject who specifies the authorization, since the same authorization can be granted by different subjects, as well as of its *timestamp*, that is, the time when it was specified. To understand why the timestamp is important in correctly implementing recursive revocation, consider the graph in Fig. 1a, which represents the authorization state for a specific privilege p on a specific object o. Nodes represent subjects, and an edge from node n_1 to node n_2 means that n_1 has granted privilege p on object o to n_2. The edge is labeled with the timestamp of the granted privilege and, optionally, with symbol "g," if the privilege has been granted with the grant option. Suppose that `Tom` revokes the authorization to `Alice`. As a result, the authorizations also held by `Matt` and `Ann` are recursively revoked because they could not have been granted if `Alice` did not receive authorization from `Tom` at time 32. In contrast, the authorization held by `Paul` is not revoked since it could have been granted even without the authorization granted by `Tom` to `Alice` at time 32, because of the privilege `Alice` had received by `Helen` at time



Access Control Administration Policies. Figure 1. Recursive revocation.

47. The authorization state resulting from the revoke operation is illustrated in Fig. 1b. Although recursive revocation has the advantage of being the most conservative solution with regard to security, it has the drawback of in some cases the unnecessarily revoking of too many authorizations. For instance, in an organization, the authorizations a user possesses are usually related to his/her job functions within the organization, rather than to his/her identity. If a user changes his/her tasks (for instance, because of a promotion), it is desirable to remove only the authorizations of the user, without revoking all the authorizations granted by the user before changing his/her job function. For this reason, research has been carried out to devise alternative semantics for the revoke operation with regard to recursive revocation. Bertino et al. [5] have proposed an alternative type of revoke operation, called *noncascading revocation*. According to this, no recursive revocation is performed upon the execution of a revoke operation. Whenever a subject revokes a privilege on an object from another subject, all authorizations which the subject may have granted using the privilege received by the revoker are not removed. Instead, they are restated as if they had been granted by the revoker.

SQL:2003 [7] adopts the object owner administration policy with delegation. A revoke request can either be issued to revoke an authorization from a subject for a particular privilege on a given object, or to revoke the authorization to play a given role. SQL:2003 supports two different options for the revoke operation. If the revoke operation is requested with the `Restrict` clause, then the revocation is not allowed if it causes the revocation of other privileges and/or the deletion of some objects from the database schema. In contrast, if the `Cascade` option is specified, then the system implements a revoke operation similar to the recursive revocation of the System R, but without taking into account authorization timestamps. Therefore, an authorization is recursively revoked only if the grantor no longer holds the grant/admin option for that, because of the requested revoke operation. Otherwise, the authorization is not deleted, regardless of the time the grantor had received the grant/admin option for that authorization. To illustrate the differences with regard to recursive revocation, consider once again Fig. 1a, and suppose that `Tom` revokes privilege *p* on object *o* to `Alice` with the `Cascade` option. With difference to the System R access control model, this revoke operation does not cause any other changes to the authorization state. The authorization granted by `Alice` to `Matt` is not deleted, because `Alice` still holds the grant option for that access (received by `Helen`).

Key Applications

Access control administration policies are fundamental in every environment where access control services are provided.

Cross-references

- ▶ Access Control
- ▶ Discretionary Access Control
- ▶ Role Based Access Control

Recommended Reading

1. Atluri V., Bertino E., Ferrari E., and Mazzoleni P. Supporting delegation in secure workflow management systems. In Proc. 17th IFIP WG 11.3 Conf. on Data and Application Security, 2003, pp. 190–202.
2. Bertino E., Bettini C., Ferrari E., and Samarati P. Decentralized administration for a temporal access control model. Inf. Syst., 22:(4)223–248, 1997.
3. Bertino E. and Ferrari E. Administration policies in a multipolicy authorization system. In Proc. 11th IFIP WG 11.3 Conference on Database Security, 1997, pp. 341–355.
4. Bertino E., Jajodia S., and Samarati P. A flexible authorization mechanism for relational data management systems. ACM Trans. Inf. Syst., 17:(2)101–140, 1999.
5. Bertino E., Samarati P., and Jajodia S. An extended authorization model. IEEE Trans. Knowl. Data Eng., 9:(1)85–101, 1997.
6. Crampton J. and Loizou G. Administrative scope: a foundation for role-based administrative models. ACM Trans. Inf. Syst. Secur., 6:(2)201–231, 2003.
7. Database Languages – SQL,ISO/IEC 9075-*, 2003.
8. Fernandez E.B., Gudes E., and Song H. A model for evaluation and administration of security in object-oriented databases. IEEE Trans. Knowl. Data Eng., 6:(2)275–292, 1994.
9. Griffiths P.P. and Wade B.W. An authorization mechanism for a relational database system. ACM Trans. Database Syst., 1:(3) 242–255, 1976.
10. Oh S., Sandhu R.S., and Zhang X. An effective role administration model using organization structure. ACM Trans. Inf. Syst. Secur., 9:(2)113–137, 2006.
11. Sandhu R.S., Bhamidipati V., and Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Trans. Inf. Syst. Secur., 2:(1)105–135, 1999.
12. Seitz L., Rissanen E., Sandholm T., Sadighi Firozabadi B., and Mulmo O. Policy Administration control and delegation using XACML and delegent. In Proc. 6th IEEE/ACM Int. Workshop on Grid Computing, 2005, pp. 49–54.
13. Zhang L., Ahn G., and Chu B. A rule-based framework for role-based delegation and revocation. ACM Trans. Inf. Syst. Secur., 6:(3)404–441, 2003.

Access Control Policy Languages

Athena Vakali
Aristotle University, Thessaloniki, Greece

Synonyms

Authorization policy languages

Definition

An access control policy language is a particular set of grammar, syntax rules (logical and mathematical), and operators which provides an abstraction-layer for access control policy specifications. Such languages combine individual rules into a single policy set, which is the basis for (user/subject) authorization decisions on accessing content (object) stored in various information resources. The operators of an access control policy language are used on attributes of the subject, resource (object), and their underlying application framework to facilitate identifying the policy that (most appropriately) applies to a given action.

Historical Background

The evolution of access control policy languages is inline with the evolving large-scale highly distributed information systems and the Internet, which turned the tasks of authorizing and controlling of accessing on a global enterprise (or on Internet) framework increasingly challenging and difficult. Obtaining a solid and accurate view of the policy in effect across its many and diverse systems and devices has guided the development of access control policy languages accordingly.

Access control policy languages followed the Digital Rights Management (DRM) standardization efforts, which had focused in introducing DRM technology into commercial and mainstream products. Originally, access control was practiced in the most popular RDBMSs by policy languages that were SQL based. Certainly, the access control policy languages evolution was highly influenced by the wide adoption of XML (late 1990s) mainly in the enterprise world and its suitability for supporting access control acts. XML's popularity resulted in an increasing need to support more flexible provisional access decisions than the initial simplistic authorization acts which were limited in an accept/deny decision. In this context, proposals of various access control policy languages were very active starting around the year 2000. This trend seemed to stabilize around 2005.

The historical pathway of such languages should highlight the following popular and general-scope access control policy languages:

- 1998: the Digital Property Rights Language (DPRL, Digital Property Rights Language, http://xml.coverpages.org/dprl.html) mostly addressed to commercial and enterprise communities was specified for describing rights, conditions, and fees to support commerce acts
- 2000: XML Access Control Language (XACL, XML Access Control Language, http://xml.coverpages.org/xacl.html) was the first XML-based access control language for the provisional authorization model
- 2001: two languages were publicized:
  - the eXtensible rights Markup Language (XrML, The Digital Rights Language for Trusted Content and Services, http://www.xrml.org/) promoted as the digital rights language for trusted content and services
  - the Open Digital Rights Language (ODRL, Open Digital Rights Language, http://odrl.net/) for developing and promoting an open standard for rights expressions for transparent use of digital content in all sectors and communities
- 2002: the eXtensible Media Commerce Language (XMCL, eXtensible Media Commerce Language, http://www.w3.org/TR/xmcl/) to communicate usage rules in an implementation-independent manner for interchange between business systems and DRM implementations
- 2003: the eXtensible Access Control Markup Language (XACML, eXtensible Access Control Markup Language, http://www.oasis-open.org/committees/xacml/) was accepted as a new OASIS, Organization for the Advancement of Structured Information Standards, http://www.oasis-open.org/, Open Standard language, designed as an XML specification with emphasis on expressing policies for information access over the Internet.
- 2005: Latest version XACML 2.0 appeared and policy languages which are mostly suited for Web services appear. These include WS-SecurityPolicy, http://www-128.ibm.com/developerworks/library/specification/ws-secpol/, which defines general security policy assertions to be applied into Web services security frameworks.

Foundations

Since Internet and networks in general are currently the core media for data and knowledge exchange, a primary issue is to assure authorized access to (protected) resources located in such infrastructures. To support access control policies and mechanisms, the use of an appropriate and suitable language is the core requirement in order to express all of the various components of access control policies, such as subjects, objects, constraints, etc. Initial attempts for expressing access control policies (consisting of authorizations) involved primary "participants" in a policy, namely the `subject` (client requesting access), the `object` (protected resource), and the `action` (right or type of access).

To understand the access control policy languages the context in which they are applied must be explained. Hence, the following notions which appear under varying terminology must be noted:

- *Content/objects*: Any physical or digital content which may be of different formats, may be divided into subparts and must be uniquely identified. Objects may also be encrypted to enable secure distribution of content.
- *Permissions/rights/actions*: Any task that will enforce permissions for accessing, using and acting over a particular content/object. They may contain constraints (limits), requirements (obligations), and conditions (such as exceptions, negotiations).
- *Subjects/users/parties*: Can be humans (end users), organizations, and defined roles which aim in consuming (accessing) content.

Under these three core entities, the policies are formed under a particular language to express offers and agreements. Therefore, the initial format of such languages authorization was (subject, object, and action) defining which subject can conduct what type of action over what object. However, with the advent of databases, networking, and distributed computing, users have witnessed (as presented in the section "Historical background") a phenomenal increase in the automation of organizational tasks covering several physical locations, as well as the computerization of information related services [6,7]. Therefore, new ideas have been added into modern access control models, like time, tasks, origin, etc. This was evident in the evolution of languages which initially supported an original syntax for policies limited in a

Access Control Policy Languages. Table 1. Summary of most popular access control policy languages

Language/ technology	Subject types	Object types	Protection granularity	Accessing core formats	Focus
DPRL/XML DTDs	Registered users	Digital XML data sources, stored on repositories	Fine-grained	Digital licenses assigned for a time-limited period	
XACL/XML syntax	Group or organization members	Particular XML documents	Fine-grained	Set of particular specified privileges	
XrML/XML schema	Registered users and/or parties	digital XML data sources	Fine-grained	Granted rights under specified conditions	
ODRL/open-source schema-valid XML syntax	Any user	Trusted or untrusted content	coarse-grained	Digital or physical rights	
XMCL/XML namespaces	Registered users	Trusted multimedia content	Coarse-grained	Specified keyword-based licenses	Particular business models
XACML/XML schema	Any users organized in categories	Domain-specific input	Fine-grained	Rule-based permissions	
WS-Security policy/ XML, SOAP	Any Web users/ Web services	Digital data sources	Fine-grained	Protection acts at SOAP Web services messages level	Web services security

3-tuple (subject, Subject primitive allows user IDs, groups, and/or role names. object, Object primitive allows granularity as fine as a single element within an XML document, and action, Action primitive consists of four kinds of actions: read, write, create, and delete.) which then was found quite simplistic and limited and it was extended to include non-XML documents, to allow roles and collections as subjects and to support more actions (such as approve, execute, etc).

Table 1 summarizes the most important characteristics of the popular general scope access control policy languages. It is evident that these languages differentiate on the subjects/users types, on the protected object/content type (which is considered as trusted when it is addressed to trusted audience/users) and on the capabilities of access control acts, which are presented under various terms and formats (rights, permissions, privileges, etc). Moreover, this table highlights the level at which the access control may be in effect for each language, i.e., the broad categorization into fine- and coarse-grained protection granularity, respectively, refers to either partitions/detailed or full document/object protection capability. Moreover, the extensibility of languages which support Web-based objects and content is noted.

To expand on the above, specific-scope languages have also emerged mainly to support research-oriented applications and tools. The most representative of such languages include:

- X-Sec [1]: To support the specification of subject credentials and security policies in Author-X and Decentral Author-X [2]. X-Sec adopts the idea of credentials which is similar to roles in that one user can be characterized by more than one credentials.

Access Control Policy Languages. Table 2. Specific-scope access control languages characteristics

	X-Sec	XACL	RBXAC	XAS syntax
Objects				
Protected resources	XML documents and DTDs	XML documents and DTDs	XML documents	XML documents and DTDs
Identification	XPath	XPath	XPath	XPath
Protection granularity	Content, attribute	Element	Content, attribute	Element
Subjects				
Identification	XML-expressed credentials	Roles, UIDs, groups	Roles	User ID, location
Grouping of subjects	No	Yes	No	Yes
Subjects hierarchy	No	Yes	Role trees	Yes
Support public subject	No	Yes	No	Yes
Policies				
Expressed in	Policy base	XACL policy file	Access control files	XAS
Closed/open	Closed	Both	Closed	Closed
Permissions/denials	Both	Both	Permissions	Both
Access modes	Authoring, browsing	Read, write, create, delete	RI, WI, RC, WC	Read
Propagation	No-prop, first-level, cascade	No/up/down	According to role tree	Local, recursive
Priority	Implicit rules	ntp, ptp, dtd	-	Hard, soft
Conflict resolution	Yes	According to priorities and implicit rules	-	Implicitly, explicitly
Other issues				
Subscription-based	Yes	Yes	Yes	Yes
Ownership	No	No	Yes	No

- XAS Syntax: Designed to support the ACP (Access Control Processor) tool [3]. It is a simplified XML-based syntax for expressing authorizations.
- RBXAC: A specification XML-based language supporting the role-based access control model [4].
- XACL: Which was originally based on a provisional authorization model and it has been designed to support ProvAuth (Provisional Authorizations) tool. Its main function is to specify security policies to be enforced upon accesses to XML documents.
- Cred-XACL [5]: A recent access control policy language focusing on credentials support on distributed systems and the Internet.

The core characteristics of these specific-scope languages are given in Table 2, which summarizes them with respect to their approach for objects and subjects management, their policies practicing and their subscription and ownership mechanisms. Such a summary is important in order to understand the "nature" of each such language in terms of objects and subjects identification, protection (sources) granularity and (subject) hierarchies, policies expression and accessing modes under prioritization, and conflict resolution constraints. Finally, it should be noted that these highlighted characteristics are important in implementing security service tasks which support several security requirements from both the system and the sources perspective.

Key Applications

Access control policy languages are involved in the transparent and innovative use of digital resources which are accessed in applications related to key nowadays areas such as publishing, distributing and consuming of electronic publications, digital images, audio and movies, learning objects, computer software and other creations in digital form.

Future Directions

From the evolution of access control policy languages, it appears that, in the future, emphasis will be given on languages that are mostly suited for Web-accessed repositories, databases, and information sources. This trend is now apparent from the increasing interest on languages that control accessing on Web services and Web data sources. At the same time, it manages the challenges posed by acknowledging and identifying users/subjects on the Web.

URL to Code

Code, examples, and application scenarios may be found for: ODRL application scenarios at http://www.w3.org/TR/odrl/#46354 and http://odrl.net/, XrML at http://www.xrml.org/, XMCL at http://www.w3.org/TR/xmcl/, XACML at http://www.oasis-open.org/committees/xacml/, and WS-SecurityPolicy at http://www-128.ibm.com/developerworks/library/specification/ws-secpol/.

Cross-references

- ▶ Access Control
- ▶ Database Security
- ▶ Role-Based Access Control
- ▶ Secure Database Development

Recommended Reading

1. Bertino E., Castano S., and Ferrari E. On specifying security policies for web documents with an XML-based language. In Proc. 6th ACM Symp. on Access Control Models and Technologies, 2001, pp. 57–65.
2. Bertino E., Castano S., and Ferrari E. Securing XML documents with author-X. IEEE Internet Computing, May–June 2001, pp. 21–31.
3. Damiani E., De Capitani di Vimercati S., Paraboschi S., and Samarati P. Desing and implementation of an access control processor for XML documents. In Proc. 9th Int. World Wide Web Conference, 2000, pp. 59–75.
4. He H. and Wong R.K. A role-based access control model for XML repositories. In Proc. 1st Int. Conf. on Web Information Systems Eng., 2000, pp. 138–145.
5. Stoupa K. Access Control Techniques in distributed systems and the Internet, Ph.D. Thesis, Aristotle University, Department of Informatics, 2007.
6. Stoupa K. and Vakali A. Policies for web security services, Chapter III. In Web and Information Security, E. Ferrari, B. Thuraisingham (eds.), Idea-Group Publishing, USA, 2006.
7. Vuong N.N., Smith G.S., and Deng Y. Managing security policies in a distributed environment using eXtensible markup language (XML). In Proc. 16th ACM Symp. on Applied Computing, 2001, pp. 405–411.

Access Methods

▶ Access Path

Access Path

Evaggelia Pitoura
University of Ioannina, Ioannina, Greece

Synonyms

Access path; Access methods

Definition

An access path specifies the path chosen by a database management system to retrieve the requested tuples from a relation. An access path may be either (i) a sequential scan of the data file or (ii) an index scan with a matching selection condition when there are indexes that match the selection conditions in the query. In general, an index matches a selection condition, if the index can be used to retrieve all tuples that satisfy the condition.

Key Points

Access paths are the alternative ways for retrieving specific tuples from a relation. Typically, there is more than one way to retrieve tuples because of the availability of indexes and the potential presence of conditions specified in the query for selecting the tuples. Typical access methods include sequential access of unordered data files (heaps) as well as various kinds of indexes. All commercial database systems implement heaps and B+ tree indexes. Most of them also support hash indexes for equality conditions.

To choose an access path, the optimizer first determines which matching access paths are available by examining the conditions specified by the query. Then, it estimates the selectivity of each access path using any available statistics for the index and data file. The *selectivity of an access path* is the number of pages (both index and data pages) accessed when the specific access path is used to retrieve the requested tuples. The access path having the smallest selectivity is called the most *selective access path*. Clearly, using the most selective access path minimizes the cost of data retrieval. Additional information can be found in [1].

Cross-references

- ▶ Index Structures for Biological Sequences
- ▶ Query Optimization
- ▶ Selectivity Estimation

Recommended Reading

1. Selinger P.G., Astrahan M.M., Chamberlin D.D., Lorie R.A., 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.

Accountability

- ▶ Auditing and Forensic Analysis

ACID Properties

Gottfried Vossen
University of Münster, Münster, Germany

Synonyms

ACID properties; Atomicity; Isolation; Consistency preservation; Durability; Persistence

Definition

The conceptual *ACID properties* (short for atomicity, isolation, consistency preservation, and durability) of a transaction together provide the key abstraction which allows application developers to disregard irregular or even malicious effects from concurrency or failures of transaction executions, as the transactional server in charge guarantees the consistency of the underlying data and ultimately the correctness of the application [1–3]. For example, in a banking context where debit/credit transactions are executed this means that no money is ever lost in electronic funds transfers and customers can rely on electronic receipts and balance statements. These cornerstones for building highly dependable information systems can be successfully applied outside the scope of online transaction processing and classical database applications as well.

Key Points

The *ACID properties* are what a database server guarantees for transaction executions, in particular in the presence of multiple concurrently running transactions and in the face of failure situations; they comprise the following four properties (whose initial letters form the word "ACID"):

Atomicity. From the perspective of a client and an application program, a transaction is executed completely or not at all, i.e., in an all-or-nothing fashion. So the effects of a program under execution on the underlying data server(s) will only become visible to the outside world or to other program executions if and when the transaction reaches its "*commit*" operation. This case implies that the transaction could be processed completely, and no errors whatsoever were discovered while it was processed. On the other hand, if the program is abnormally terminated before reaching its commit operation, the data in the underlying data servers will be left in or automatically brought back to the state in which it was before the transaction started, i.e., the data appears as if the transaction had never been invoked at all.

Consistency preservation: Consistency constraints that are defined on the underlying data servers (e.g., keys, foreign keys) are preserved by a transaction; so a transaction leads from one consistent state to another. Upon the commit of a transaction, all integrity constraints defined for the underlying database(s) must be satisfied; however, between the beginning and the end of a transaction, inconsistent intermediate states are tolerated and may even be unavoidable. This property generally cannot be ensured in a completely automatic manner. Rather, it is necessary that the application is programmed such that the code between the beginning and the commit of a transaction will eventually reach a consistent state.

Isolation: A transaction is isolated from other transactions, i.e., each transaction behaves as if it was operating alone with all resources to itself. In particular, each transaction will "see" only consistent data in the underlying data sources. More specifically, it will see only data modifications that result from committed transactions, and it will see them only in their entirety, and never any effects of an incomplete transaction. This is the decisive property that allows to hide the fallacies and pitfalls of concurrency from the application developers. A sufficient condition for isolation is that concurrent executions are equivalent to sequential ones, so that all transactions appear as if they were executed one after the other rather than in an interleaved manner; this condition is made precise through serializability.

Durability: When the application program from which a transaction derives is notified that the transaction has been successfully completed (i.e., when the commit point of the transaction has been reached), all updates the transaction has made in the underlying data servers are guaranteed to survive subsequent software or hardware failures. Thus, updates of committed transactions are durable (until another transaction later modifies the same data items) in that they persist even across failures of the affected data server(s).

Therefore, a transaction is a set of operations executed on one or more data servers which are issued by an application program and are guaranteed to have the ACID properties by the runtime system of the involved servers. The "ACID contract" between the application program and the data servers requires the program to demarcate the boundaries of the transaction as well as the desired outcome – successful or abnormal termination – of the transaction, both in a dynamic manner. There are two ways a transaction can finish: it can commit, or it can abort. If it commits, all its changes to the database are installed, and they will remain in the database until some other application makes further changes. Furthermore, the changes will seem to other programs to take place together. If the transaction aborts, none of its changes will take effect, and the DBMS will rollback by restoring previous values to all the data that was updated by the application program. A programming interface of a transactional system consequently needs to offer three types of calls: (i) "*begin transaction*" to specify the beginning of a transaction, (ii) "*commit transaction*" to specify the successful end of a transaction, and (iii) "*rollback transaction*" to specify the unsuccessful end of a transaction with the request to abort the transaction.

The core requirement for a transactional server is to provide the ACID guarantees for sets of operations that belong to the same transaction issued by an application program requires that the server. This requires a *concurrency control* component to guarantee the isolation properties of transactions, for both committed and aborted transactions, and a *recovery* component to guarantee the atomicity and durability of transactions. The server may or may not provide explicit support for consistency preservation. In addition to the ACID contract, a transactional server should meet a number of technical requirements: A transactional data server (which most often will be a database system) must provide *good performance* with a given hardware/software configuration, or more generally, a good cost/performance

ratio when the configuration is not yet fixed. Performance typically refers to the two metrics of *high throughput*, which is defined as the number of successfully processed transactions per time unit, and of *short response times*, where the response time of a transaction is defined as the time span between issuing the transaction and its successful completion as perceived by the client.

While the ACID properties are crucial for many applications in which the transaction concept arises, some of them are too restrictive when the transaction model is extended beyond the read/write context. For example, business processes can be cast into various forms of *business transactions*, i.e., long-running transactions for which atomicity and isolation are generally too strict. In these situations, additional or alternative guarantees need to be employed.

Cross-references

- ▶ Atomicity
- ▶ Concurrency Control
- ▶ Extended Transaction Models
- ▶ Multi-Level Recovery and the ARIES Algorithm
- ▶ Serializability
- ▶ Snapshot Isolation
- ▶ SQL Isolation Levels
- ▶ Transaction Model

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, MA, 1987.
2. Bernstein P.A. and Newcomer E. Principles of Transaction Processing for the Systems Professional. Morgan Kaufmann, San Francisco, CA, 1997.
3. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Francisco, CA, 1993.

ACID Transaction

- ▶ Transaction

Acquisitional Query Languages

- ▶ Database Languages for Sensor Networks

Active and Real-Time Data Warehousing

Mukesh Mohania[1], Ullas Nambiar[1], Michael Schrefl[2], Millist Vincent[3]
[1]IBM India Research Lab, New Delhi, India
[2]University of Linz, Linz, Austria
[3]University of South Australia, Adelaide, SA, Australia

Synonyms

Right-time data warehousing

Definition

Active Data Warehousing is the technical ability to capture transactions when they change, and integrate them into the warehouse, along with maintaining batch or scheduled cycle refreshes. An active data warehouse offers the possibility of automating routine tasks and decisions. The active data warehouse exports decisions automatically to the On-Line Transaction Processing (OLTP) systems.

Real-time Data Warehousing describes a system that reflects the state of the warehouse in real time. If a query is run against the real-time data warehouse to understand a particular facet about the business or entity described by the warehouse, the answer reflects the state of that entity at the time the query was run. Most data warehouses have data that are highly latent – or reflects the business at a point in the past. A real-time data warehouse has low latency data and provides current (or real-time) data.

Simply put, a real-time data warehouse can be built using an active data warehouse with a very low latency constraint added to it. An alternate view is to consider active data warehousing as being a design methodology suited to tactical decision-making based on very current data while real-time data warehousing is a collection of technologies that refresh a data warehouse frequently. A real-time data warehouse is one that acquires, cleanses, transforms, stores, and disseminates information in real time. An active data warehouse, on the other hand, operates in a non-real-time response mode with one-or-more OLTP systems.

Historical Background

A *data warehouse* is a decision support database that is periodically updated by extracting, transforming, and loading operational data from several OLTP databases.

In the data warehouse, OLTP data is arranged using the (multi) dimensional data modeling approach (see [1] for a basic approach and [2] for details on translating an OLTP data model into a dimensional model), which classifies data into *measures* and *dimensions*. In recent years, several multidimensional data models have been proposed [3–6]. An in-depth comparison is provided by Pedersen and Jensen in [5]. The basic unit of interest in a data warehouse is a *measure* or *fact* (e.g., sales), which represent countable, semisummable, or summable information concerning a business process. An instance of a measure is called *measure value*. A measure can be analyzed from different perspectives, which are called the *dimensions* (e.g., location, product, time) of the data warehouse [7]. A dimension consists of a set of *dimension levels* (e.g., time: Day, Week, Month, Quarter, Season, Year, ALLTimes), which are organized in multiple hierarchies or *dimension paths* [6] (e.g., Time[Day] → Time[Month] → Time[Quarter] → Time[Year] → Time[ALLTimes]; Time[Day] → Time [Week] → Time[Season] → Time[ALLTimes]). The hierarchies of a dimension form a lattice having at least one top dimension level and one bottom dimension level. The measures that can be analyzed by the same set of dimensions are described by a *base cube* or *fact table*. A base cube uses level instances of the lowest dimension levels of each of its dimensions to identify a measure value. The relationship between a set of measure values and the set of identifying level instances is called *cell*. Loading data into the data warehouse means that new *cells* will be added to *base cubes* and new level instances will be added to *dimension levels*. If a dimension D is related to a measure m by means of a base cube, then the hierarchies of D can be used to aggregate the *measure values* of m using operators like SUM, COUNT, or AVG. Aggregating measure values along the hierarchies of different dimensions (i.e., *rollup*) creates a multidimensional view on data, which is known as *data cube* or *cube*. Deaggregating the measures of a cube to a lower dimension level (i.e., *drilldown*) creates a more detailed cube. Selecting the subset of a cube's cells that satisfy a certain selection condition (i.e., *slicing*) also creates a more detailed cube.

The data warehouses are used by analysts to find solutions for decision tasks by using OLAP (On-Line Analytical Processing) [7] systems. The decision tasks can be split into three, viz. *non-routine, semi-routine*, and *routine*. Non-routine tasks occur infrequently and/or do not have a generally accepted decision criteria. For example, strategic business decisions such as introducing a new brand or changing an existing business policy are non-routine tasks. Routine tasks, on the other hand, are well structured problems for which generally accepted procedures exist and they occur frequently and at predictive intervals. Examples can be found in the areas of product assortment (change price, withdraw product, etc.), customer relationship management (grant loyalty discounts etc.), and in many administrative areas (accept/reject paper based on review scores). Semi-routine tasks are tasks that require a non-routine solution – e.g., paper rated contradictory must be discussed by program committee. Since, most tasks are likely to be routine, it is logical to automate processing of such tasks to reduce the delay in decision-making.

Active data warehouses [8] were designed to enable data warehouses to support automatic decision-making when faced with routine decision tasks and routinizable elements of semi-routine decision tasks. The active data warehouse design extends the technology behind active database systems. Active database technology transforms passive database systems into reactive systems that respond to database and external events through the use of rule processing features [9,10]. Limited versions of active rules exist in commercial database products [11,12].

Real-time data warehousing captures business activity data as it occurs. As soon as the business activity is complete and there is data about it, the completed activity data flows into the data warehouse and becomes available instantly. In other words, real-time data warehousing is a framework for deriving information from data as the data becomes available. Traditionally, data warehouses were regarded as an environment for analyzing historic data, either to understand what has happened or simply to log the changes as they happened. However, of late, businesses want to use them to predict the future: e.g., to predict customers likely to churn; and thereby seek better control of the business. However, until recently, it was not practical to have *zero-latency* data warehouses – the process of extracting data had too much of an impact on the source systems concerned, and the various steps needed to cleanse and transform the data required multiple temporary tables and took several hours to run. However, the increased visibility of (the value of) warehouse data, and the take-up by a wider audience within the organization, has lead to a number

of product developments by IBM [13], Oracle [14], and other vendors that make real-time data warehousing now possible.

Foundations

The two example scenarios below describe typical situations in which active rules can be used to automate decision-making:

Scenario 1: Reducing the price of an article. Twenty days after a soft drink has been launched on a market, analysts compare the quantities sold during this period with a standardized indicator. This indicator requires the total quantities sold during the 20-day period do not drop below a threshold of 10,000 sold items. If the analyzed sales figures are below this threshold, the price of the newly launched soft drink will be reduced by 15.

Scenario 2 : Withdrawing articles from a market. At the end of every quarter, high-priced soft drinks which are sold in Upper Austrian stores will be analyzed. If the sales figures of a high-priced soft drink have continuously dropped, the article will be withdrawn from the Upper Austrian market. Analysts inspect sales figures at different granularities of the time dimension and at different granularities of the location dimension. Trend, average, and variance measures are used as indicators in decision-making.

Rules that mimic the analytical work of a business analyst are called *analysis rules* [8]. The components of analysis rules constitute the *knowledge model* of an active data warehouse (and also a real-time data warehouse). The knowledge model determines *what* an analyst must consider when he specifies an active rule to automate a routine decision task.

An analysis rule consists of (i) the *primary dimension level* and (ii) the *primary condition*, which identify the objects for which decision-making is necessary, (iii) the *event*, which triggers rule processing, (iv) the *analysis graph*, which specifies the cubes for analysis, (v) the *decision steps*, which represent the conditions under which a decision can be made, and (vi) the *action*, which represents the rule's decision task. Below is a brief description of the components of an analysis rule. Detailed discussion is given in [8].

Event: Events are used to specify the timepoints at which analysis rules should be carried out. Active data warehouses provide three kinds of events: (i) OLTP method events, (ii) relative temporal events, and (iii) calendar events. OLTP method events describe basic happenings in the data warehouse's sources. Relative temporal events are used to define a temporal distance between such a basic happening and carrying out an analysis rule. Calendar events represent fixed points in time at which an analysis rule may be carried out. Structurally, every event instance is characterized by an occurrence time and by an event identifier. In its event part, an analysis rule refers to a calendar event or to a relative temporal event.

An *OLTP method event* describes a happening in the data warehouse's source systems that is of interest to analysis rules in the active data warehouse. Besides occurrence time and event identifier, the attributes of an OLTP method event are a reference to the dimension level for which the OLTP method event occurred and the parameters of the method invocation. To make OLTP method events available in data warehouses, a data warehouse designer has to define the schema of OLTP method events and extend the data warehouse's extract/transform/load mechanism. Since instances of OLTP method events are loaded some time after their occurrence, analysis rules cannot be triggered directly by OLTP method events.

Temporal events determine the timepoints at which decision-making has to be initiated. Scenario 1 uses the relative temporal event "twenty days after launch" while Scenario 2 uses the periodic temporal event "end of quarter." The *conditions* for decision-making are based on indicators, which have been established in manual decision-making. Each condition refers to a multidimensional cube and therefore "analyzing" means to evaluate the condition on this cube. Scenario 1 uses a quantity-based indicator, whereas scenario 2 uses value-based indicators for decision-making. The decision whether to carry out the rule's *action* depends on the result of evaluating the conditions. The action of scenario 1 is to reduce the price of an article, whereas the action of scenario 2 is to withdraw an article from a market.

Primary Condition: Several analysis rules may share the same OLTP method as their action. These rules may be carried out at different timepoints and may utilize different multidimensional analyses. Thus, a certain analysis rule usually analyzes only a subset of the level instances that belong to the rule's primary dimension level. The primary condition is used to determine for a level instance of the primary dimension level whether multidimensional analysis should be carried out by the analysis rule. The primary condition is specified as a Boolean expression, which refers to the

describing attributes of the primary dimension level. If omitted, the primary condition evaluates to TRUE.

Action: The purpose of an analysis rule is to automate decision-making for objects that are available in OLTP systems and in the data warehouse. A *decision* means to invoke (or not to invoke) a method on a certain object in an OLTP system. In its action part, an analysis rule may refer to a single OLTP method of the primary dimension level, which represents a transaction in an OLTP system. These methods represent the *decision space* of an active data warehouse. To make the transactional behavior of an OLTP object type available in the active data warehouse, the data warehouse designer must provide (i) the specifications of the OLTP object type's methods together with required parameters, (ii) the preconditions that must be satisfied before the OLTP method can be invoked in the OLTP system, and (iii) a conflict resolution mechanism, which solves contradictory decisions of different analysis rules. Since different analysis rules can make a decision for the same level instance of the rules' primary dimension level during the same active data warehouse cycle, a *decision conflict* may occur. Such conflicts are considered as interrule conflicts. To detect interrule conflicts, a *conflict table* covering the OLTP methods of the decision space is used. The tuples of the conflict table have the form $<m1, m2, m3>$, where $m1$ and $m2$ identify two conflicting methods and $m3$ specifies the conflict resolution method that will be finally executed in OLTP systems. If a conflict cannot be solved automatically it has to be reported to analysts for manual conflict resolution.

Analysis Graph: When an analyst queries the data warehouse to make a decision, he or she follows an incremental topdown approach in creating and analyzing cubes. Analysis rules follow the same approach. To automate decision-making, an analysis rule must "know" the cubes that are needed for multidimensional analysis. These cubes constitute the *analysis graph*, which is specified once by the analyst. The n dimensions of each cube of the analysis graph are classified into one *primary dimension*, which represents the level instances of the primary dimension level, and $n - 1$ *analysis dimensions*, which represent the multidimensional space for analysis. Since a level instance of the primary dimension level is described by one or more cells of a cube, multidimensional analysis means to compare, aggregate, transform, etc., the measure values of these cells. Two kinds of multidimensional analysis are carried out at each cube of the analysis graph: (i) select the level instances of the primary dimension level whose cells comply with the decision-making condition (e.g., withdraw an article if the sales total of the last quarter is below USD 10,000) and (ii) select the level instances of the primary dimension level whose cells comply with the condition under which more detailed analysis (at finer grained cubes) are necessary (e.g., continue analysis if the sales total of the last quarter is below USD 500,000). The multidimensional analysis that is carried out on the cubes of the analysis graph are called decision steps. Each decision step analyzes the data of exactly one cube of the analysis graph. Hence, analysis graph and decision steps represent the knowledge for multidimensional analysis and decision-making of an analysis rule.

Enabling real-time data warehousing: As mentioned earlier, real-time data warehouses are active data warehouses that are loaded with data having (near) zero latency. Data warehouse vendors have used multiple approaches such as *hand-coded scripting* and data extraction, transformation, and loading (ETL) [15] solutions to serve the data acquisition needs of a data warehouse. However, as users move toward real-time data warehousing, there is a limited choice of technologies that facilitate real-time data delivery. The challenge is to determine the right technology approach or combination of solutions that best meets the data delivery needs. Selection criteria should include considerations for frequency of data, acceptable latency, data volumes, data integrity, transformation requirements and processing overhead. To solve the real-time challenge, businesses are turning to technologies such as enterprise application integration (EAI) [16] and transactional data management (TDM) [17], which offer high-performance, low impact movement of data, even at large volumes with sub-second speed. EAI has a greater implementation complexity and cost of maintenance, and handles smaller volumes of data. TDM provides the ability to capture transactions from OLTP systems, apply mapping, filtering, and basic transformations and delivers to the data warehouse directly. A more detailed study of the challenges involved in implementing a real-time data warehouse is given in [18].

Key Applications

Active and Real-time data warehouses enable businesses across all industry verticals to gain competitive advantage by allowing them to run *analytics* solutions over the

most recent data of interest that is captured in the warehouse. This will provide them with the ability to make intelligent business decisions and better understand and predict customer and business trends based on accurate, up-to-the-second data. By introducing real-time flows of information to data warehouses, companies can increase supply chain visibility, gain a complete view of business performance, and increase service levels, ultimately increasing customer retention and brand value.

The following are some additional business benefits of active and real-time data warehousing:

- *Real-time Analytics:* Real-time analytics is the ability to use all available data to improve performance and quality of service at the moment they are required. It consists of dynamic analysis and reporting, right at the moment (or very soon after) the resource (or information) entered the system. In a practical sense, real time is defined by the need of the consumer (business) and can vary from a few seconds to few minutes. In other words, more frequent than daily can be considered real-time, because it crosses the overnight-update barrier. With increasing availability of active and real-time data warehouses, the technology for capturing and analyzing real-time data is increasingly becoming available. Learning how to apply it effectively becomes the differentiator. Implementing real-time analytics requires the integration of a number of technologies that are not interoperable off-the-shelf. There are no established best practices. Early detection of fraudulent activity in financial transactions is a potential environment for applying real-time analytics. For example, credit card companies monitor transactions and activate counter measures when a customer's credit transactions fall outside the range of expected patterns. However, being able to correctly identify fraud while not offending a well-intentioned valuable customer is a critical necessity that adds complexity to the potential solution.
- *Maximize ERP Investments:* With a real-time data warehouse in place, companies can maximize their Enterprise Resource Planning (ERP) technology investment by turning integrated data into business intelligence. ETL solutions act as an integral bridge between ERP systems that collect high volumes of transactions and business analytics to create data reports.
- *Increase Supply Chain Visibility:* Real-time data warehousing helps streamline supply chains through highly effective business-to-business communications and identifies any weak links or bottlenecks, enabling companies to enhance service levels and gain a competitive edge.
- *Live 360° View of Customers:* The active database solutions enable companies to capture, transform, and flow all types of customer data into a data warehouse, creating one seamless database that provides a 360° view of the customer. By tracking and analyzing all modes of interaction with a customer, companies can tailor new product offerings, enhance service levels, and ensure customer loyalty and retention.

Future Directions

Data warehousing has greatly matured as a technology discipline; however enterprises that undertake data warehousing initiatives continue to face fresh challenges that evolve with the changing business and technology environment. Most future needs and challenges will come in the areas of active and real-time data warehousing solutions. Listed below are some future challenges:

- *Integrating Heterogeneous Data Sources:* The number of enterprise data sources is growing rapidly, with new types of sources emerging every year. Enterprises want to integrate the unstructured data generated from customer emails, chat and voice call transcripts, feedbacks, and surveys with other internal data in order to get a complete picture of their customers and integrate internal processes. Other sources for valuable data include ERP programs, operational data stores, packaged and homegrown analytic applications, and existing data marts. The process of integrating these sources into a data warehouse can be complicated and is made even more difficult when an enterprise merges with or acquires another enterprise.
- *Integrating with CRM tools:* Customer relationship management (CRM) is one of the most popular business initiatives in enterprises today. CRM helps enterprises attract new customers and develop loyalty among existing customers with the end result of increasing sales and improving profitability. Increasingly, enterprises want to use the holistic view of the customer to deliver value-added services to

the customer based on her overall value to the enterprise. This would include, automatically identifying when an important life event is happening and sending out emails with necessary information and/or relevant products, gauging the mood of the customer based on recent interactions, and alerting the enterprise before it is too late to retain the customer and most important of all identifying customers who are likely to accept suggestions about upgrades of existing products/services or be interested in newer versions. The data warehouse is essential in this integration process, as it collects data from all channels and customer touch points, and presents a unified view of the customer to sales, marketing, and customer-care employees. Going forward, data warehouses will have to provide support for analytics tools that are embedded into the warehouse, analyze the various customer interactions continuously, and then use the insights to trigger *actions* that enable delivery of the above-mentioned value-added services. Clearly, this requires an active data warehouse to be tightly integrated with the CRM systems. If the enterprise has low latency for insight detection and value-added service delivery then a real-time data warehouse would be required.

- *In-built data mining and analytics tools:* Users are also demanding more sophisticated business intelligence tools. For example, if a telecom customer calls to cancel his call-waiting feature, real-time analytic software can detect this and trigger a special offer of a lower price in order to retain the customer. The need is to develop a new generation of data mining algorithms that work over data warehouses that integrate heterogeneous data and have self-learning features. These new algorithms must automate data mining and make it more accessible to mainstream data warehouse users by providing explanations with results, indicating when results are not reliable and automatically adapting to changes in underlying predictive models.

Cross-references

- ▶ Cube Implementations
- ▶ Data Warehouse Interoperability
- ▶ Data Warehousing Systems: Foundations and Architectures
- ▶ ETL
- ▶ Multidimensional Modeling
- ▶ On-Line Analytical Processing
- ▶ Query Processing in Data Warehouses
- ▶ Transformation

Recommended Reading

1. Kimball R. and Strethlo K. Why decision support fails and how to fit it. ACM SIGMOD Rec., 24(3):91–97, 1995.
2. Golfarelli M., Maio D., and Rizzi S. Conceptual design of data warehouses from E/R schemes. In Proc. 31st Annual Hawaii Int. Conf. on System Sciences, Vol. VII. 1998, pp. 334–343.
3. Lehner W. Modeling large scale OLAP scenarios. In Advances in Database Technology, Proc. 6th Int. Conf. on Extending Database Technology, 1998, pp. 153–167.
4. Li C. and Wang X.S. A data model for supporting on-line analytical processing. In Proc. Int. Conf. on Information and Knowledge Management, 1996, pp. 81–88.
5. Pedersen T.B. and Jensen C.S. Multidimensional data modeling for complex data. In Proc. 15th Int. Conf. on Data Engineering, 1999, pp. 336–345.
6. Vassiliadis P. Modeling multidimensional databases, cubes and cube operations. In Proc. 10th Int. Conf. on Scientific and Statistical Database Management, 1998, 53–62.
7. Chaudhuri S. and Dayal U. An overview of data warehousing and OLAP technology. ACM SIGMOD Rec., 26(1):65–74, 1997.
8. Thalhammer T., Schrefl M., and Mohania M. Active data warehouses: complementing OLAP with analysis rules. Data Knowl. Eng., 39(3):241–269, 2001.
9. ACT-NET Consortium. The active database management system manifesto: a rulebase of ADBMS featueres. ACM SIGMOD Rec., 25(3), 1996.
10. Simon E. and Dittrich A. Promises and realities of active database systems. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 642–653.
11. Brobst S. Active data warehousing: a new breed of decision support. In Proc. 13th Int. Workshop on Data and Expert System Applications, 2002, pp. 769–772.
12. Borbst S. and Rarey J. The five stages of an active data warehouse evolution. Teradata Mag., 38–44, 2001.
13. IBM DB2 Data Warehouse Edition. http://www-306.ibm.com/software/data/db2/dwe/.
14. Rittman M. Implementing Real-Time Data Warehousing Using Oracle 10g. Dbazine.com. http://www.dbazine.com/datawarehouse/dw-articles/rittman5.
15. Kimball R. and Caserta J. The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data. Wiley, 2004.
16. Linthicum R.S. Enterprise Application Integration. Addison-Wesley, 1999.
17. Improving SOA with Goldengate TDM Technology. GoldenGate White Paper, October 2007.
18. Langseth J. Real-Time Data Warehousing: Challenges and Solutions. DSSResources.COM, 2004.
19. Paton N.W. and Diaz O. Active Database Systems. ACM Comput. Surv., 1(31),1999.

Active Database, Active Database (Management) System

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Definition

An active database (aDB) or active database (management) system (aDBS/aDBMS) is a database (management) system that supports reactive behavior through ECA-rules.

Historical Background

The term active database was first used in the early 1980s [12]. Some related active database work was also done within the area of expert database systems in the mid 1980s, but it was not until the mid/late 1980s that the research on supporting ECA rules in database systems took off, for example [10,12,18]. During the 1990s the area was extensively explored through more than twenty suggested active database prototypes and a large body of publications:

- Seven workshops were held between 1993 and 1997: RIDS [12,16,17], RIDE-ADS [20], Dagstuhl Seminar [5] and ARTDB [3,4].
- Two special issues of journals [8,9] and one special issue of ACM Sigmod Record [1].
- Two text books [13,19] and one ACM Computing Survey paper [15].

In addition, the groups within the ACT-NET consortium (A European research network of Excellence on active databases 1993–1996) reached a consensus on what constitutes an active database management system with the publication of the Active Database System Manifesto [2].

Most of the active databases are monolithic and assume a centralized environment, consequently, the majority of the prototype implementations do not consider distributed issues. Initial work on how active databases are affected by distributed issues are reported in [7].

Foundations

An active database can automatically react to events such as database transitions, time events, and external signals in a timely and efficient manner. This is in contrast to traditional database systems, which are passive in their behaviors, so that they only execute queries and transactions when they are explicitly requested to do so. Previous approaches to support reactive behavior can broadly be classified into:

- Periodically polling the database.
- Embedding or encoding event detection and related action execution in the application code.

The first approach implies that the queries must be run exactly when the event occurs. The frequency of polling can be increased in order to detect such an event, but if the polling is too frequent, then the database is overloaded with queries and will most often fail. On the other hand, if the frequency is too low, the event will be missed.

The second approach implies that every application which updates the database needs to be augmented with condition checks in order to detect events. For example, an application may be extended with code to detect whether the quantity of certain items has fallen below a given level. From a software engineering point of view, this approach is inappropriate, since a change in a condition specification implies that every application that uses the modified condition needs to be updated.

Neither of the two previous approaches can satisfactorily support reactive behavior in a database context [10]. An active database system avoids the previous disadvantages by moving the support for reactive behavior inside the database (management) system. Reactive behavior in an active database is supported by ECA-rules that have the following semantics: when an event is detected, evaluate a condition, and if the condition is true, execute an action.

Similar to describing an object by its static features and dynamic features, an active database can be described by its knowledge model (static features) and execution model (dynamic features). Thus, by investigating the knowledge model and execution model of an active database, one can identify what type of ECA rules that can be defined and how the active database behave at run-time.

Key Applications

An aDB or aDBS/aDBMS is useful for any non-mission critical application that require reactive behavior.

Future Directions

Looking back, the RIDS'97 workshop marks the end of the active database period, since there are very few

active database publications after 1997. However, the concept of ECA-rules has resurfaced and has been picked up by other research communities such as Complex Event Processing and Semantic Web. In contrast to typical active database approaches that assume a centralized environment, the current research on ECA rules within Complex Event Processing and Semantic Web assume that the environment is distributed and heterogeneous. Thus, as suggested within the REWERSE project [3], one cannot assume that the event, condition, and action parts of an ECA rule are defined in one single ECA rule language. For example, the event part of an ECA-rule can be defined in one language (e.g., Snoop), whereas the condition part and action part are defined in a completely different rule language.

The popularity of using XML for manipulating data has also led to proposals of ECA rule markup languages. These ECA rule markup languages are used for storing information about ECA rules and facilitates exchange of ECA-rules between different rule engines and applications.

One research question that remains from the active database period is how to model and develop applications that use ECA rules. Some research on modeling ECA rules has been carried out, but there is no widely agreed approach for modeling ECA rules explicitly in UML, or how to derive ECA rules from existing UML diagrams.

Cross-references

- ▶ Active Database Execution Model
- ▶ Active Database Knowledge Model
- ▶ Complex Event Processing
- ▶ ECA Rules

Recommended Reading

1. ACM SIGMOD Record. Special Issue on Rule Management and Processing in Expert Databases, 1989.
2. ACT-NET Consortium The active database management system manifesto: a rulebase of ADBMS features. ACM SIGMOD Rec., 25(3):40–49, 1996.
3. Alferes J.J., Amador R., and May W. A general language for evolution and reactivity in the semantic Web. In Proc. 3rd Workshop on Principles and Practice of Semantic Web Reasoning, 2005, pp. 101–115.
4. Andler S.F. and Hansson J. (eds.). In Proc. 2nd International Workshop on Active, Real-Time, and Temporal Database Systems, LNCS, vol. 1553, Springer, 1998.
5. Berndtsson M. and Hansson J. Workshop report: the first international workshop on active and real-time database systems. SIGMOD Rec., 25(1):64–66, 1996.
6. Buchmann A., Chakravarthy S., and Dittrich K. Active Databases. Dagstuhl Seminar No. 9412, Report No. 86, 1994.
7. Bültzingsloewen G., Koschel A., Lockemann P.C., and Walter H.D. ECA Funtionality in a Distributed Environment. Monographs in Computer Science, chap. 8, Springer, 1999, pp. 147–175.
8. Chakravarthy S. (ed.), Special Issue on Active Databases, vol. 15, IEEE Quarterly Bulletin on Data Engineering, 1992.
9. Chakravarthy S. and Widom J. (eds.), Special Issue on the Active Database Systems, Journal of Intelligent Information Systems 7, 1996.
10. Dayal U., Blaustein B., Buchmann A., et al. S.C. HiPAC: A Research Project in Active, Time-Constrained Database Management. Tech. Rep. CCA-88-02, Xerox Advanced Information Technology, Cambridge, 1988.
11. Dittrich K.R., Kotz A.M., and Mulle J.A. An Event/Trigger Mechanism to Enforce Complex Consistency Constraints in Design Databases. ACM SIGMOD Rec., 15(3):22–36, 1986.
12. Geppert A. and Berndtsson M. (eds.). Proc. 3rd International Workshop on Rules in Database Systems, LNCS, vol. 1312, Springer, 1997.
13. Morgenstern M. Active Databases as a Paradigm for Enhanced Computing Environments. In Proc. 9th Int. Conf. on Very Data Bases, 1983, pp. 34–42.
14. Paton N.W. (ed.) Active Rules in Database Systems. Monographs in Computer Science, Springer, 1999.
15. Paton N.W. and Diaz O. Active Database Systems. ACM Comput. Surv, 31(1):63–103, 1999.
16. Paton N.W. and Williams M.W. (eds.). In Proc. 1st International Workshop on Rules in Database Systems, Springer, Berlin, 1994.
17. Sellis T. (ed.). In Proc. 2nd International Workshop on Rules in Database Systems, vol. 905, Springer, 1995.
18. Stonebraker M., Hearst M., and Potamianos S. Commentary on the POSTGRES Rules System. SIGMOD Rec., 18(3):5–11, 1989.
19. Widom J. and Ceri S. (eds.) Active Database Systems: Triggers and Rules For Advanced Database Processing. Morgan Kaufmann, 1996.
20. Widom J. and Chakravarthy S. (eds.). In Proc. 4th International Workshop on Research Issues in Data Engineering – Active Database Systems, 1994.

Active Database Management System Architecture

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Synonyms

ADBMS infrastructure; ADBMS framework; ADBMS

Definition

The active database management system (ADBMS) architecture is the software organization of a DBMS

with active capabilities. That is, the architecture defines support for active capabilities expressed in terms of services, significant components providing the services as well as critical interaction among these services.

Historical Background

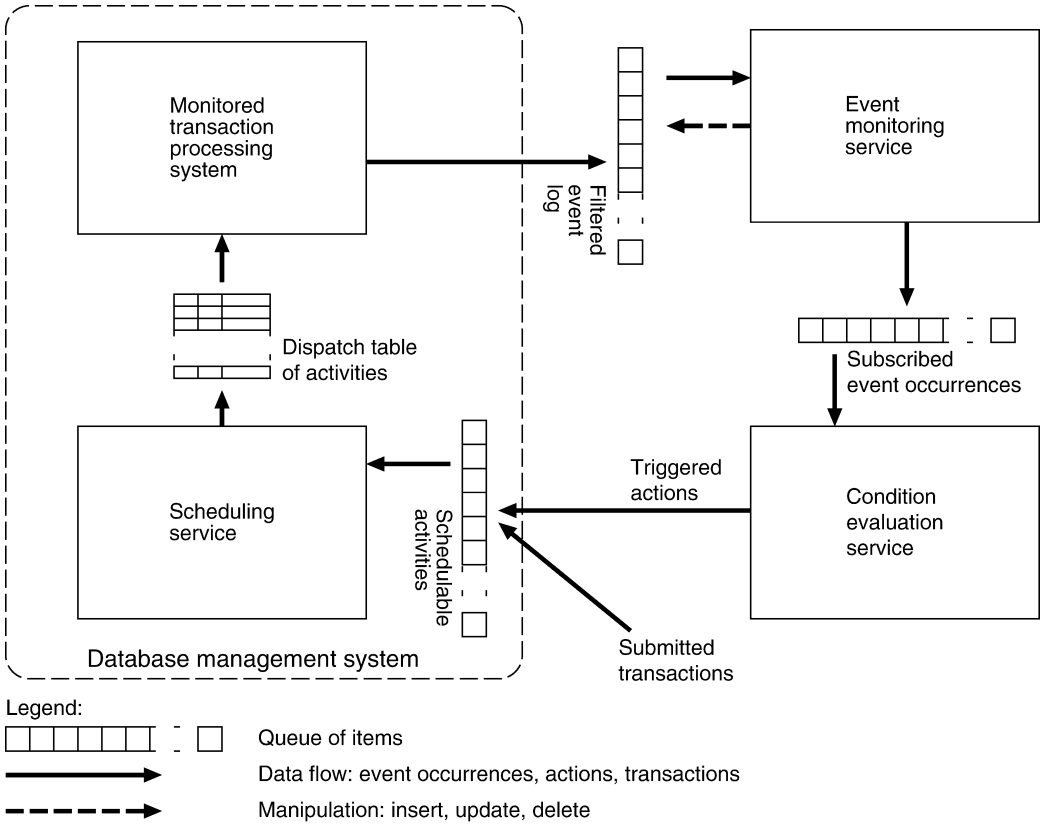
Several architectures has been proposed: HiPAC [5,8], REACH [4], ODE [14], SAMOS [10], SMILE [15], and DeeDS [1]. Each of these architectures emphasize particular issues concerning the actual DBMS that they are based on as well as the type of support for active capabilities. Paton and Diaz [18] provide an excellent survey on this topic. Essentially, these architectures propose that the *active capabilities* of an ADBMS require the services specified in Table 1. It is assumed that queries to the database are encompassed in transactions and hence transactions imply queries as well as database manipulation operations such as insertion, updates and deletion of tuples.

The services in Table 1 interact as depicted in Fig. 1. Briefly, transactions are submitted to the scheduling service that updates the dispatch table read by the transaction processing service. When these transactions are processed by the transaction processing service events are generated. These events are signaled to the event monitoring service that analyzes them. Events that are associated with rules (subscribed events) are signaled to the rule evaluation service that evaluates the conditions of *triggered rules* (i.e., rules associated with signaled events). The actions of the rules whose conditions are true are submitted for scheduling and are executed as dictated by the scheduling policy. These actions execute as part of some transaction according to the coupling mode and can, in turn, generate events. This is a general description of the service interaction and it can be optimized by refining it for a specific purpose, for example, in immediate coupling mode no queues between the services are actually needed.

In more detail, transactions are submitted to the scheduling service via a queue of schedulable activities; this queue of schedulable activities is processed and a dispatch table of schedulable activities is updated. This scheduling service encompasses scheduling of transactions as well as ECA rule actions in addition to other necessary schedulable activities. It is desirable for the scheduling service to encompass all these types of schedulable activities, because they impact each other, since they compete for the same resources. The next step in the processing chain is the monitored transaction processing service, which includes the transaction management, lock management, and log management [11, Chap. 5], as well as a database query engine (cf. query processor [20, Chap. 1]), but not the scheduling service. Another way to view the transaction processing service is as a passive database management system without the transaction scheduling service. The transaction processing service is denoted "monitored," since it generates events that are handled by the active capabilities. The monitored transaction processing service executes transactions and ECA rule actions according to the dispatch table. When transactions execute, event occurrences are signaled to the event monitoring service via a filtered event log. When event monitoring executes, it updates the filtered event log and submits subscribed events to the rule evaluation service. An example of event log filtering is that if a composite event occurrence is detected, then for optimization reasons (cf. dynamic programming) this event occurrence is stored in the filtered event log. Another example is that when events are no longer needed, then they are pruned; for example, when a transaction is aborted, then all event occurrences can be pruned unless intertransaction events are allowed (implying that dirty reads may occur). The rule evaluation service reads the

Active Database Management System Architecture. Table 1. Services in active database management systems

Service	Responsibility
Event monitoring	The event monitoring service is responsible for collecting events, analyzing events and disseminating results of the analysis (in terms of events) to subscribers, in particular, ECA rules.
Rule evaluation	The rule evaluation service is responsible for invoking condition evaluation of triggered ECA rules and submit actions for execution to the scheduler.
Scheduling service	The scheduling service is responsible for readying and ordering schedulable activities such as ECA rule actions, transactions etc. for execution.



Active Database Management System Architecture. Figure 1. Service interaction view of architecture (based on architecture by Paton and Diaz [18]).

queue of subscribed events, finds the triggered rules and evaluates their conditions. These conditions may be queries, logical expressions or arbitrary code depending on the active database system [9]. The rule evaluation results in a set of actions that is submitted to the scheduling service for execution.

The general view of active capabilities (in Fig. 1) can be refined and implemented in different ways. As mentioned, it is possible to optimize an implementation by removing the queues between the services if only immediate coupling mode is considered; this result in less overhead, but restricts the expressibility of ECA-rules significantly. A service can be implemented via one or more servers. These servers can be replicated to different physical nodes for performance or dependability reasons (e.g., availability, reliability).

In active databases, a set of issues have a major impact on refinement and implementation of the general service-oriented view depicted in Fig. 1. These issues are: (i) coupling modes; (ii) interaction with typical database management services such as transaction management, lock management, recovery management (both pre-crash such as logging and checkpointing and post-crash such as the actually recovery) (cf., for example, transaction processing by Gray and Reuter [11, Chap. 4]); (iii) when and how to invoke services; and (iv) active capabilities in distributed active databases.

The coupling modes control how rule evaluation is invoked in response to events and how the ECA rule actions are submitted, scheduled, dispatched and executed for rules whose conditions are true (see entry "Coupling modes" for more detailed description). There are different alternatives to interaction with a database system. One alternative is to place active database services on top of existing database management systems. However, this is problematic if the database management system is not extended with active

capabilities [4]. For example, the deferred coupling mode require that when a transaction is requested to commit, then queued actions should be evaluated. This requires that the transaction management to interact with the rule evaluation and scheduling services during commit processing (e.g., by using back hooks in the database management system). Further, to be useful, the detached coupling mode has a set of significant varieties [4] that require the possibility to express constraints between transactions.

The nested transaction model [16] is a sound basis for active capabilities. For example, deferred actions can be executed as subtransactions that can be committed or aborted independently of the parent transaction. Nested transactions still require that existing services are modified. Alternatively rule evaluation can be performed as subtransactions.

To achieve implicit events the database schema translation process needs to automatically instrument the monitored systems. An inferior solution is to extend an existing schema with instrumented entities, for example, each class in an object-oriented database can be inherited to an instrumented class. In this example, there is no way to enforce that the instrumented classes are actually used. The problem is to modify the database schema translation process, since this is typically an intrinsic part in commercial DBMSs.

Concerning issue (iii), the services must be allocated to existing resources and scheduled together with the transactions. Typically, the services are implemented as a set of server processes and transactions are performed by transaction programs running as processes (cf., [11]). These processes are typically scheduled, dispatched and executed as a response to the requests from outside the database management system or as a direct or indirect response to a timeout. Each service is either invoked when something is stored in the queue or table or explicitly invoked, for example, when the system clock is updated to reflect the new time. The issues concerning scheduling are paramount in any database management system for real-time system purposes [2].

Event monitoring can either be (i) implicitly invoked whenever an event occurs, or it can be (ii) explicitly invoked. This is similar to coupling modes, but it is between the event sources (e.g., transaction processing service and application) and the event monitoring service rather than in between the services of the active capabilities. Case (i) is prevalent in most active database research, but it has a negative impact in terms of determinism of the result of event monitoring. For example, the problem addressed in the event specification and event detection entry concerning the unintuitive semantics of the disjunctive event operator is a result of implicit invocation. In distributed and real-time systems, explicit invocation is preferable in case (ii), since it provides the operating system with the control when something should be evaluated. Explicit invocation solves the problem of disjunction operator (see event specification and event detection entries), since the event expressions defining composite event types can be explicitly evaluated when all events have been delivered to event monitoring rather than implicitly evaluated whenever an event is delivered.

In explicit invocation of event monitoring, the different event contexts can be treated in different ways. For example, in recent event context, only the most recent result is of interest in implicit invocation. However, in terms of explicit invocation, all possible most recent event occurrences may be of interest, not only the last one. For example, it may be desirable to keep the most recent event occurrence per time slot rather than per terminating event.

Issue (iv) has been addressed in, for example, DeeDS [1], COBEA [15], Hermes [19], X2TS [5]. Further, it has been addressed in event based systems for mobile networks by Mühl et al. [17]. Essentially, it is necessary to perform event detection in a moving time window, where the end of the time window is the current time. All events that are older than the beginning of the time window can be removed and ignored. Further, the heterogeneity must be addressed and there are XML-based solutions (e.g., Common Base Events [6]).

Another issue that is significant in distributed active databases is the time and order of events. For example, in Snoop [8] it is suggested to separate global and local event detection, because of the difference in the time granularity of the local view of time and the global (distributed) view of time.

Foundations

For a particular application domain, common significant requirements and properties as well as pre-requisites of

available resources need to be considered to refine the general architecture. Depending on the requirements, properties and pre-requisites, different compromises are reached. One example is the use of composite event detection in active real-time databases. In REACH [4], composite event detection is disallowed for real-time transactions. The reason for this is that during composite event detection, contributing events are locked and this locking affects other transaction in a harmful way with respect to meeting deadlines. A different approach is proposed in DeeDS [1], where events are stored in the database and cached in a special filtered event log; during event composition, events are not locked thus enabling the use of composite event detection for transaction with critical deadlines. The cost is that isolation of transactions can be violated unless it is handled by the active capabilities.

Availability is an example of a property that significantly affects the software architecture. For example, availability is often considered significant in distributed systems; that is, even though physical nodes may fail, communications links may be down, or the other physical nodes may be overloaded, one should get, at least, some defined level of service from the system. An example of availability requirements is that emergency calls in phone switches should be prioritized over non-emergency calls, a fact that entails that existing phone call connections can be disconnected to let an emergency call through. Another example to improve availability is pursued in DeeDS [1], where eventual consistency is investigated as a mean to improve availability of data. The cost is that data can temporarily be inconsistent.

As addressed in the aforementioned examples, different settings affect the architecture. Essentially, there are two approaches that can be mixed: (i) refine or invent new method, tools, techniques to solve a problem, and these method, tools, techniques can stem from different but relevant research areas; (ii) refine the requirements or pre-requisites to solve the problem (e.g., weaken the ACID properties of transactions).

Key Applications

The architecture of ADBMSs is of special interest to developers of database management systems and their applications. In particular, software engineering issues are of major interest. Researchers performing experiments can make use of this architecture to enable valid experiments, study effects of optimizations etc.

Concerning real examples of applications, only simple things such as using rules for implementing alerters, for example, when an integrity constraint is violated. SQL Triggers implement simple ECA rules in immediate coupling mode between event monitoring and rule evaluation as well as between rule evaluation and action execution.

Researchers have aimed for various application domains such as:

- Stock market
- Inventory control
- Bank applications

Essentially, any application domain in which there is an interest to move functionality from the applications to the database schema to reduce the interdependence between applications and databases.

Future Directions

There are no silver bullets in computer science or software engineering and each refinement of the architecture (in Fig. 1) is a compromise providing or enabling certain features and properties. For example, by allowing only detached coupling mode it is easier to achieve timeliness, an important property of real-time systems; however, the trade-off is that it is difficult to specify integrity rules in terms of ECA-rules, since the integrity checks are performed in a different transaction. The consequence is that dirty transactions as well as compensating transactions that perform recovery from violated integrity rules must be allowed.

It is desirable to study architectures addressing how to meet specific requirement of the application area (e.g., accounting information in mobile ad-hoc networks), the specific environment in which the active database are used (e.g., distributed systems, real-time systems, mobile ad-hoc networks, limited resource equipment). The major criteria for a successful architecture (e.g., by refining an existing architecture) is if anyone can gain something from using it. For example, Borr [3] reported that by refining their architecture by employing transaction processing they improved productivity, reliability as well as average throughput in their heterogenous distributed reliable applications.

An area that has received little attention in active database is optimization of processing. For example, how can queries to the database be optimized with condition evaluation if conditions are expressed as

arbitrary queries? Another question is how to group actions to optimize performance? So far, the emphasis has been on expressibility as well as techniques how to enable active support in different settings. Another area that has received little attention is recovery processing, both pre-crash and post-crash recovery. For example, how should recovery with respect to detached but dependent transactions be managed?

Intertransaction events and rules has been proposed by, for example, Buchmann et al. [4]. How should this be managed with respect to the isolation levels proposed by Gray and Reuter [11, Chap. 7]?

There are several other areas with which active database technology can be combined. Historical examples include real-time databases, temporal databases, main-memory databases, geographical information systems. One area that has received little attention is how enable reuse of database schemas.

Cross-references

- ▶ Active Database Coupling Modes
- ▶ Active Database Execution Model
- ▶ Active Database Knowledge Model
- ▶ Event Detection
- ▶ Event Specification

Recommended Reading

1. Andler S., Hansson J., Eriksson J., Mellin J., Berndtsson M., and Eftring B. DeeDS Towards a Distributed Active and Real-Time Database System. ACM SIGMOD Rec., 25(1), 1996.
2. Berndtsson M. and Hansson J. Issues in active real-time databases. In Proc. 1st Int. Workshop on Active and Real-Time Database System, 1995, pp. 142–150.
3. Borr A.J. Robustness to crash in a distributed database: A non shared-memory multi-processor approach. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 445–453.
4. Buchmann A.P., Zimmermann J., Blakeley J.A., and Wells D.L. Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 117–128.
5. Chakravarthy S., Blaustein B., Buchmann A.P., Carey M., Dayal U., Goldhirsch D., Hsu M., Jauhuri R., Ladin R., Livny M., McCarthy D., McKee R., and Rosenthal A. HiPAC: A Research Project In Active Time-Constrained Database Management. Tech. Rep. XAIT-89-02, Xerox Advanced Information Technology, 1989.
6. Common Base Events. Http://www.ibm.com/developerworks/library/specification/ws-cbe/.
7. Chakravarthy S., Krishnaprasad V., Anwar E., and Kim S.K. Composite Events for Active Database: Semantics, Contexts, and Detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
8. Dayal U., Blaustein B., Buchmann A., Chakravarthy S., Hsu M., Ladin R., McCarty D., Rosenthal A., Sarin S., Carey M.J., Livny M., and Jauharu R. The HiPAC Project: Combining active databases and timing constraints. ACM Sigmod Rec., 17(1), 1988.
9. Eriksson J. Real-Time and Active Databases: A Survey. In Proc. 2nd Int. Workshop on Active, Real-Time, and Temporal Database Systems, 1997, pp. 1–23.
10. Gatziu S. Events in an Active Object-Oriented Database System. Ph.D. thesis, University of Zurich, Switzerland, 1994.
11. Gray J. and Reuter A. Transaction processing: Concepts and techniques. Morgan Kaufmann, Los Altos, CA, 1994.
12. Jaeger U. Event Detection in Active Databases. Ph.D. thesis, University of Berlin, 1997.
13. Liebig C.M. and Malva A.B. Integrating Notifications and Transactions: Concepts and X2TS Prototype. In Second International Workshop on Engineering Distributed Objects, 2000, pp. 194–214.
14. Lieuwen D.F., Gehani N., and Arlein R. The ODE active database: Trigger semantics and implementation. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 412–420.
15. Ma C. and Bacon J. COBEA: A CORBA-based Event Architecture. In Proc. 4th USENIX Conf. Object-Oriented Technologies and Syst., 1998, pp. 117–132.
16. Moss J.E.B. Nested transactions: An approach to reliable distributed computing. MIT, 1985.
17. Mühl G., Fiege L., and Pietzuch P.R. Distributed event-based systems. Springer, Berlin, 2006.
18. Paton N. and Diaz O. Active database systems. ACM Comput. Surv., 31(1):63–103, 1999.
19. Pietzuch P. and Bacon J. Hermes: A Distributed Event-Based Middleware Architecture. In Proc. 22nd Int. Conf. on Distributed Computing Systems Workshop. Vienna, Austria, 2002, pp. 611–618.
20. Ullman J.D. Principles of Database Systems. Computer Science, 1982.

Active Database Coupling Modes

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Definition

Coupling modes specify execution points for ECA rule conditions and ECA rule actions with respect to the triggering event and the transaction model.

Historical Background

Coupling modes for ECA rules were first suggested in the HiPAC project [2,3].

Foundations

Coupling modes are specified for event-condition couplings and for condition-action couplings. In detail, the event-condition coupling specifies when the condition should be evaluated with respect to the triggering event, and the condition-action coupling specifies when the rule action should be executed with respect to the evaluated rule condition (if condition is evaluated to true).

The three most common coupling modes are: immediate, deferred, and decoupled. The immediate coupling mode preempts the execution of the transaction and immediately initiates condition evaluation and action execution. In the deferred coupling mode, condition evaluation and action execution is deferred to the end of the transaction (before transaction commit). Finally, in decoupled (also referred to as detached) coupling mode, condition evaluation and action execution is performed in separate transactions.

Specifying event-condition couplings and condition-action couplings in total isolation from each other is not a good idea. What first might seem to be one valid coupling mode for event-condition and one valid coupling mode for condition-action, can be an invalid coupling mode when used together. Thus, when combining event-condition couplings and condition-action couplings, not all combinations of coupling modes are valid. The HiPAC project [2,3] proposed seven valid coupling modes, see Table 1.

- *Immediate, immediate*: the rule condition is evaluated immediately after the event, and the rule action is executed immediately after the rule condition.
- *Immediate, deferred*: the rule condition is evaluated immediately after the event, and the execution of the rule action is deferred to the end of the transaction.
- *Immediate, decoupled*: the rule condition is evaluated immediately after the event, and the rule action is decoupled in a totally separate and parallel transaction.
- *Deferred, deferred*: both the evaluation of the rule condition and the execution of the rule action is deferred to the end of the transaction.
- *Deferred, decoupled*: the evaluation of the rule condition is deferred to the end of the transaction, and the rule action is decoupled in a totally separate and parallel transaction.
- *Decoupled, immediate*: the rule condition is decoupled in a totally separate and parallel transaction, and the rule action is executed (in the same parallel transaction) immediately after the rule condition.
- *Decoupled, decoupled*: the rule condition is decoupled in a totally separate and parallel transaction, and the rule action is decoupled in another totally separate and parallel transaction.

The two invalid coupling modes are:

- *Deferred, immediate*: this combination violates the semantics of ECA rules. That is, rule conditions must be evaluated before rule actions are executed. One cannot preempt the execution of the transaction immediately after the event and execute the rule action and at the same time postpone the condition evaluation to the end of the transaction.
- *Decoupled, deferred*: this combination violates transaction boundaries. That is, one cannot decouple the condition evaluation in a separate and parallel transaction and at the same time postpone the

Active Database Coupling Modes. Table 1. Coupling modes

	Condition-Action		
Event-Condition	*Immediate*	*Deferred*	*Decoupled*
Immediate	condition evaluated and action executed after event	condition evaluated after event, action executed at end of transaction	condition evaluated after event, action executed in a separate transaction
Deferred	not valid	condition evaluated and action executed at end of transaction	condition evaluated at end of transaction, action executed in a separate transaction
Decoupled	in a separate transaction: condition evaluated and action executed after event	not valid	condition evaluated in one separate transaction, action executed in another separate transaction

execution of the rule action to the end of the original transaction, since one cannot know when the condition evaluation will take place. Thus, there is a risk that the action execution in the original transaction will run before the condition has been evaluated in the parallel transaction.

Rule actions executed in decoupled transactions can either be dependent upon or independent of the transaction in which the event took place.

The research project REACH (REal-time ACtive Heterogeneous System) [1] introduced two additional coupling modes for supporting side effects of rule actions that are irreversible. The new coupling modes are variants of the detached casually dependent coupling mode: sequential casually dependent, and exclusive casually dependent. In sequential casually dependent, a rule is executed in a separate transaction. However, the rule execution can only begin once the triggering transaction has committed. In exclusive casually dependent, a rule is executed in a detached parallel transaction and it can commit only if the triggering transaction failed.

Cross-references

► Active Database Execution Model
► ECA-rules

Recommended Reading

1. Branding H., Buchmann A., Kudrass T., and Zimmermann J. Rules in an Open System: The REACH Rule System. In Proc. 1st International Workshop on Rules in Database Systems, Workshops in Computing, 1994, pp. 111–126.
2. Dayal U., Blaustein B.A., Buchmann S.C., et al. The HiPAC project: Combining active databases and timing constraints. ACM SIGMOD Rec., 17(1):51–70, 1988.
3. Dayal U., Blaustein B., Buchmann A., Chakravarthy S., and et al. HiPAC: A Research Project in Active, Time-Constrained Database Management. Tech. Rep. CCA-88-02, Xerox Advanced Information Technology, Cambridge, 1988.

Active Database Execution Model

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Definition

The execution model of an active database describes how a set of ECA rules behave at run time.

Key Points

The execution model describes how a set of ECA rules (i.e., active database rulebase) behave at run time [2,4]. Any execution model of an active database must have support for: (i) detecting event occurrences, (ii) evaluating conditions, and (iii) executing actions.

If an active database supports composite event detection, it needs a policy that describes how a composite event is computed. A typical approach is to use the event consumption modes as described in Snoop [1]: recent, chronicle, continuous, and cumulative. In the recent event context, only the most recent constituent events will be used to form composite events. In the chronicle event context, events are consumed in chronicle order. The earliest unused initiator/terminator pair are used to form the composite event. In the continuous event context, each initiator starts the detection of a new composite event and a terminator may terminate one or more composite event occurrences. The difference between continuous and chronicle event contexts is that in the continuous event context, one terminator can detect more than one occurrence of the composite event. In the cumulative event context, all events contributing to a composite event are accumulated until the composite event is detected. When the composite event is detected, all contributing events are consumed. Another approach to these event consumption modes is to specify a finer semantics for each event by using logical events as suggested in [3].

Once an event has been detected, there are several execution policies related to rule conditions and rule actions that must be in place in the execution model. Thus an execution model for an active database should provide answers to the following questions [2,4,5]:

- When should the condition be evaluated and when should the action should be executed with respect to the triggering event and the transaction model? This is usually specified by coupling modes.
- What happens if an event triggers several rules?
  - Are all rules evaluated, a subset, or only one rule?
  - Are rules executed in parallell, according to rule priority, or non-deterministically?
- What happens if one's rules trigger another set of rules?
  - What happens if the rule action of one rule negates the rule condition of an already triggered rule?
  - Can cycles appear? For example, can a rule trigger itself?

The answers to the above questions are important to know, as they dictate how a ECA rule system will behave at run time. If the answers to the above questions are not known, then the behavior of the ECA rule application becomes unpredictable.

Cross-references

- ▶ Active Database Coupling Modes
- ▶ Active Database Rulebase
- ▶ Composite Event
- ▶ Database Trigger
- ▶ ECA Rules

Recommended Reading

1. Chakravarthy S., Krishnaprasad V., Anwar E., and Kim S.K. Composite Events for Active Databases: Semantics Contexts and Detection. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 606–617.
2. Dayal U., Blaustein B., Buchmann A., and Chakravarthy S. et al. HiPAC: A Research Project in Active, Time-Constrained Database Management. Technical Report CCA-88-02, Xerox Advanced Information Technology, Cambridge, 1988.
3. Gehani N., Jagadish H.V., and Smueli O. Event specification in an active object-oriented database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1992, pp. 81–90.
4. Paton N.W. and Diaz O. Active Database Systems. ACM Comput. Surv, 31(1):63–103, 1999.
5. Widom J. and Finkelstein S. Set-Oriented Production Rules in Relational Database Systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 259–270.

Active Database Knowledge Model

Mikael Berndtsson, Jonas Mellin
University of Skövde, Skövde, Sweden

Definition

The knowledge model of an active database describes what can be said about the ECA rules, that is what type of events are supported, what type of conditions are supported, and what type of actions are supported?

Key Points

The knowledge model describes what types of events, conditions, and actions that are supported in an active database. Another way to look at the knowledge model is to imagine what type of features are available in an ECA rule definition language.

A framework of dimensions for the knowledge model is presented in [3]. Briefly, each part of an ECA rule is associated with dimensions that describe supported features. Thus, an event can be described as either a primitive event or a composite event, how it was generated (source), whether the event is generated for all instances in a given set or only for a subset (event granularity), what type (if event is a composite event) of operators and event consumption modes are used in the detection of the composite event.

Conditions are evaluated against a database state. There are three different database states that a rule condition can be associated with [3]: (i) the database state at the start of the transaction, (ii) the database state when the event was detected, and (iii) the database state when the condition is evaluated.

There are four different database states that a rule action can be associated with [3]: (i) the database state at the start of the transaction, (ii) the database state when the event was detected, and (iii) the database state when the condition is evaluated, and (iv) the database state just before action execution. The type of rule actions range from internal database updates (e.g., update a table) to external programs (e.g., send email).

Within the context of the knowledge model it is also useful to consider how ECA rules are represented, for example inside classes, as data members, or first class objects. Representing ECA rules as first class objects [1,2] is a popular choice, since rules can be treated as any other object in the database and traditional database operations can be used to manipulate the ECA rules. Thus, representing ECA rules as first class objects implies that ECA rules are not dependent upon the existence of other objects.

The knowledge model of an active database should also describe whether the active database supports passing of parameters between the ECA rule parts, for example passing of parameters from the event part to the condition part.

Related to the knowledge model is the execution model that describes how ECA rules behave at run time.

Cross-references

- ▶ Active Database Execution Model
- ▶ ECA Rules

Recommended Reading

1. Dayal U., Blaustein B., Buchmann A. et al. S.C. HiPAC: A Research Project in Active, Time-Constrained Database Management. Tech. Rep. CCA-88-02, Xerox Advanced Information Technology, Cambridge, 1988.

2. Dayal U., Buchmann A., and McCarthy D. Rules are objects too: a knowledge model for an active, object-oriented database system. In Proc. 2nd Int. Workshop on Object-Oriented Database Systems, 1988, pp. 129–143.
3. Paton N.W. and Diaz O. Active database systems. ACM Comput. Surv., 31(1):63–103, 1999.

Active Database Rulebase

AnnMarie Ericsson[1], Mikael Berndtsson[2], Jonas Mellin[3]
University of Skövde, Skövde, Sweden

Definition
An active database rulebase is a set of ECA rules that can be manipulated by an active database.

Key Points
An active database rulebase is a set of ECA rules that can be manipulated by an active database. Thus, an ADB rulebase is not static, but it evolves over time. Typically, ECA rules can be added, deleted, modified, enabled, and disabled. Each update of the ADB rulebase can potentially lead to different behaviors of the ECA rules at run time, in particular with respect to termination and confluence.

Termination concerns whether a set of rules is guaranteed to terminate. A set of rules may have a non-terminating behavior if rules are triggering each other in a circular order, for example, if the execution of rule R1 triggers rule R2 and the execution of rule R2 triggers rule R1. A set of rules is confluent if the outcome of simultaneously triggered rules is unique and independent of execution order.

Cross-references
► ECA Rules

Active Databases

► Event Driven Architecture

Active Disks

► Active Storage

Active Document

► Active XML

Active Storage

Kazuo Goda
The University of Tokyo, Tokyo, Japan

Synonyms
Active Disks; Intelligent Disks

Definition
Active Storage is a computer system architecture which utilizes processing power in disk drives to execute application code. Active Storage was introduced in separate academic papers [1–3] in 1998. The term Active Storage is sometimes identified merely with the computer systems proposed in these papers. Two synonyms, Active Disk and Intelligent Disk, are also used to refer to Active Storage. The basic idea behind Active Storage is to offload computation and data traffic from host computers to the disk drives themselves such that the system can achieve significant performance improvements for data intensive applications such as decision support systems and multimedia applications.

Key Points
A research group at Carnegie Mellon University proposed, in [3], a storage device called Active Disk, which has the capability of downloading application-level code and running it on a processor embedded on the device. Active Disk has a performance advantage for I/O bound scans, since processor-per-disk processing can potentially reduce data traffic on interconnects to host computers and yield great parallelism of scans. E. Riedel et al. carefully studied the potential benefits of using Active Disks for four types of data intensive applications, and introduced analytical performance models for comparing traditional server systems and Active Disks. They also prototyped ten Active Disks, each having a DEC Alpha processor and two Seagate disk drives, and demonstrated almost linear scalability in the experiments.

A research group at University of California at Berkeley discussed a vision of Intelligent Disks (IDISKs) in [2]. The approach of Intelligent Disk is similar to that

of Active Disk. K. Keeton et al. carefully studied the weaknesses of shared-nothing clusters of workstations and then explored the possibility of replacing the cluster nodes with Intelligent Disks for large-scale decision support applications. Intelligent Disks assumed higher complexity of applications and hardware resources in comparison with CMU's Active Disks.

Another Active Disk was presented by a research group at the University of California at Santa Barbara and University of Maryland in [1]. A. Acharya et al. carefully studied programming models to exploit disk-embedded processors efficiently and safely and proposed algorithms for typical data intensive operations such as selection and external sorting, which were validated by simulation experiments.

These three works are often recognized as opening the gate for new researches of Intelligent Storage Systems in the post-"database machines" era.

Cross-references

- ▶ Database Machine
- ▶ Intelligent Storage Systems

Recommended Reading

1. Acharya A., Mustafa U., and Saltz J.H. Active disks: programming model, algorithms and evaluation. In Proc. 8th Int. Conf. Architectural Support for Programming Lang. and Operating Syst., 1998, pp. 81–91.
2. Keeton K., Patterson D.A., and Hellerstein J.M. A case for intelligent disks (IDISKs). SIGMOD Rec., 27(3):42–52, 1998.
3. Riedel E., Gibson G.A., and Faloutsos C. Active storage for large-scale data mining and multimedia. In Proc. 24th Int. Conf. on Very Large Data Bases, 1998, pp. 62–73.

Active XML

Serge Abiteboul[1], Omar Benjelloun[2], Tova Milo[3]
[1]INRIA, Saclay Île-de-France, Orsay, Cedex, France
[2]Google Inc., Mountain view, CA, USA
[3]Tel Aviv University, Tel Aviv, Israel

Synonyms

Active document; AXML

Definition

Active XML documents (AXML documents, for short) are XML documents [12] that may include embedded calls to Web services [13]. Hence, AXML documents are a combination of regular "extensional" XML data with data that is defined "intensionally," i.e., as a description that enables obtaining data dynamically (by calling the corresponding service).

AXML documents evolve in time when calls to their embedded services are triggered. The calls may bring data once (when invoked) or continually (e.g., if the called service is a continuous one, such as a subscription to an RSS feed). They may even update existing parts of the document (e.g., by refreshing previously fetched data).

Historical Background

The AXML language was originally proposed at INRIA around 2002. Work around AXML has been going there in the following years. A survey of the research on AXML is given in [13]. The software, primarily under the form of an AXML system, is available as open source software. Resources on Active XML may be found on the project's Web site [11].

The notion of embedding function calls into data is old. Embedded functions are already present in relational systems as stored procedures. Of course, method calls form a key component of object databases. For the Web, scripting languages such as PHP or JSP have made popular the integration of processing inside HTML or XML documents. Combined with standard database interfaces such as JDBC and ODBC, functions are used to integrate results of (SQL) queries. This idea can also be found in commercial software products, for instance, in Microsoft Office XP, SmartTags inside Office documents can be linked to Microsoft's .NET platform for Web services.

The originality of the AXML approach is that it proposed to *exchange* such documents, building on the fact that Web services may be invoked from anywhere. In that sense, this is truly a language for distributed data management. Another particularity is that the logic (the AXML language) is a subset of the AXML algebra.

Looking at the services in AXML as queries, the approach can be viewed as closely related to recent works based on XQuery [14] where the query language is used to describe query plans. For instance the DXQ project [7] developed at ATT and UCSD emphasizes the distributed evaluation of XQuery queries. Since one can describe documents in an XQquery syntax, such approaches encompass in

some sense AXML documents where the service calls are XQuery queries.

The connection with deductive databases is used in [1] to study the diagnosis problems in distributed networks. A similar approach is followed in [8] for declarative network routing.

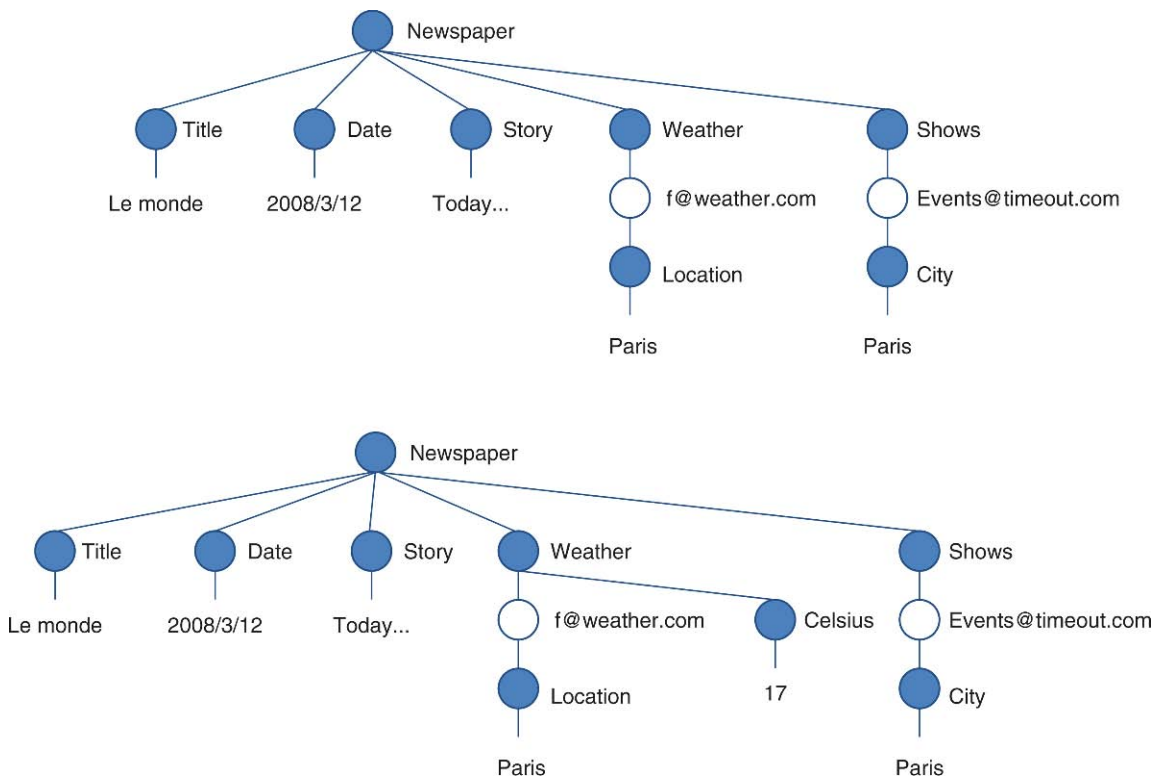
It should be observed that the AXML approach touches upon most database areas. In particular, the presence of intensional data leads to views, deductive databases and data integration. The activation of calls contained in a document essentially leads to active databases. AXML services may be activated by external servers, which relates to subscription queries and stream databases. Finally, the evolution of AXML documents and their inherent changing nature lead to an approach of workflows and service choreography in the style of business artifacts [10].

The management of AXML document raises a number of issues. For instance, the evaluation of queries over active documents is studied in [2]. The "casting" of a document to a desired type is studied in [9]. The distribution of documents between several peers and their replication is the topic of [4].

Foundations

An AXML document is a (syntactically valid) XML document, where service calls are denoted by special XML elements labeled *call*. An example AXML document is given in Fig. 1. The figure shows first the XML serialized syntax, then a more abstract view of the same document as a labeled tree. The document in the figure describes a (simplified) newspaper homepage consisting of (i) some extensional information (the name of the newspaper, the current date, and a news story), and (ii) some intensional information (service calls for the weather forecast, and for the current exhibits). When the services are called, the tree evolves. For example, the tree at the bottom is what results from a call to the service *f* at *weather.com* to obtain the temperature in Paris.

AXML documents fit nicely in a peer-to-peer architecture, where each peer is a persistent store of AXML



Active XML. Figure 1. An AXML document.

documents, and may act both as a client, by invoking the service calls embedded in its AXML documents, and as a server, by providing Web services over these documents.

Two fundamental issues arise when dealing with AXML documents. The first one is related to the exchange of AXML documents between peers, and the second one is related to query evaluation over such data.

Documents Exchange: When exchanged between two applications/peers, AXML documents have a crucial property: since Web services can be called from anywhere on the Web, data can either be materialized before sending, or sent in its intensional form and left to the receiver to materialize if and when needed. Just like *XML Schemas* do for standard XML, *AXML schemas* let the user specify the desired format of the exchanged data, including which parts should remain intensional and which should be materialized. Novel algorithms allow the sender to determine (statically or dynamically) which service invocations are required to "cast" the document to the required data exchange format [9].

Query evaluation: Answering a query on an AXML document may require triggering some of the service calls it contains. These services may, in turn, query other AXML documents and trigger some other services, and so on. This recursion, based on the management of intensional data, leads to a framework in the style of *deductive databases*. Query evaluation on AXML data can therefore benefit from techniques developed in deductive databases such as Magic Sets [6]. Indeed, corresponding AXML query optimization techniques where proposed in [1,2].

Efficient query processing is, in general, a critical issue for Web data management. AXML, when properly extended, becomes an algebraic language that enables query processors installed on different peers to collaborate by exchanging streams of (A)XML data [14]. The crux of the approach is (i) the introduction of generic services (i.e., services that can be provided by several peers, such as query processing) and (ii) some explicit control of distribution (e.g., to allow delegating part of some work to another peer).

Key Applications

AXML and the AXML algebra target all distributed applications that involve the management of distributed data. AXML is particularly suited for data integration (from databases and other data resources exported as Web services) and for managing (active) views on top of data sources. In particular, AXML can serve as a formal foundation for mash-up systems. Also, the language is useful for (business) applications based on evolving documents in the style of business artifacts, and on the exchange of such information. The fact that the exchange is based on flows of XML messages makes it also well-adapted to the management of distributed streams of information.

Cross-references

- ▶ Active Document
- ▶ BPEL
- ▶ Web Services
- ▶ W3C XML Query Language
- ▶ XML
- ▶ XML Types

Recommended Reading

1. Abiteboul S., Abrams Z., and Milo T. Diagnosis of Asynchronous Discrete Event Systems – Datalog to the Rescue! In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 358–367.
2. Abiteboul S., Benjelloun O., Cautis B., Manolescu I., Milo T., and Preda N. Lazy Query Evaluation for Active XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 227–238.
3. Abiteboul S., Benjelloun O., and Milo T. The Active XML project, an overview, VLDB J, 17(5):1019–1040, 2008.
4. Abiteboul S., Bonifati A., Cobena G., Manolescu I., and Milo T. Dynamic XML Documents with Distribution and Replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 527–538.
5. Abiteboul S., Manolescu I., and Taropa E. A Framework for Distributed XML Data Management. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006.
6. Bancilhon F., Maier D., Sagiv Y., and Ullman J.D. Magic Sets and Other Strange Ways to Implement Logic Programs. In Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1986, pp. 1–15.
7. DXQ: Managing Distributed System Resources with Distributed XQuery. http://db.ucsd.edu/dxq/.
8. Loo B.T., Condie T., Garofalakis M., Gay D.E, Hellerstein J.M., Maniatis P., Ramakrishnan R., Roscoe T., and Stoica I. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 97–108.
9. Milo T., Abiteboul S., Amann B., Benjelloun O., and Ngoc F.D. Exchanging Intensional XML data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 289–300.
10. Nigam A. and Caswell N.S. Business artifacts: an approach to operational specification. IBM Syst. J., 42(3):428–445, 2003.
11. The Active XML homepage. http://www.activexml.net/.

12. The Extensible Markup Language (XML) 1.0 (2nd edn). http://www.w3.org/TR/REC-xml.
13. The W3C Web Services Activity. http://www.w3.org/2002/ws.
14. The XQuery language. http://www.w3.org/TR/xquery.

Activity

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms
Step; Node; Task; Work element

Definition
A description of a piece of work that forms one logical step within a process. An activity may be a manual activity, which does not support computer automation, or a workflow (automated) activity. A workflow activity requires human and/or machine resources to support process execution; where human resource is required an activity is allocated to a workflow participant.

Key Points
A process definition generally consists of many process activities which are logically related in terms of their contribution to the overall realization of the business process.

An activity is typically the smallest unit of work which is scheduled by a workflow engine during process enactment (e.g., using transition and pre/post-conditions), although one activity may result in several work items being assigned (to a workflow participant).

Wholly manual activities may form part of a business process and be included within its associated process definition, but do not form part of the automated workflow resulting from the computer supported execution of the process.

An activity may therefore be categorized as "manual," or "automated." Within this document, which is written principally in the context of workflow management, the term is normally used to refer to an automated activity.

Cross-references
► Activity Diagrams
► Actors/Agents/Roles
► Workflow Model

Activity Diagrams

Luciano Baresi
Politecnico di Milano University, Milan, Italy

Synonyms
Control flow diagrams; Object flow diagrams; Flowcharts; Data flow diagrams

Definition
Activity diagrams, also known as control flow and object flow diagrams, are one of the UML (unified modeling language [11]) behavioral diagrams. They provide a graphical notation to define the sequential, conditional, and parallel composition of lower-level behaviors. These diagrams are suitable for business process modeling and can easily be used to capture the logic of a single use case, the usage of a scenario, or the detailed logic of a business rule. They model the workflow behavior of an entity (system) in a way similar to *state diagrams* where the different activities are seen as the states of doing something. Although they could also model the internal logic of a complex operation, this is not their primary use since tangled operations should always be decomposed into simpler ones [1,2].

An activity [3] represents a behavior that is composed of individual elements called *actions*. Actions have incoming and outgoing edges that specify control and data flow from and to other nodes. Activities may form invocation hierarchies invoking other activities, ultimately resolving to individual actions.

The execution of an activity implies that each contained action be executed zero, one, or more times depending on the execution conditions and the structure of the activity. The execution of an action is initiated by the termination of other actions, the availability of particular objects and data, or the occurrence of external events. The execution is based on token flow (like Petri Nets). A token contains an object, datum, or locus of control, and is present in the activity diagram at a particular node. When an action begins execution, tokens are accepted from some or all of its input edges and a token is placed on the node. When an action completes execution, a token is removed from the node and tokens are moved to some or all of its output edges.

Historical Background

OMG (Object Management Group, [10]) proposed and standardized *activity diagrams* by borrowing concepts from flow-based notations and some formal methods. As for the first class, these diagrams mimic *flowcharts* [6] in their idea of step-by-step representation of algorithms and processes, but they also resemble data and control flow diagrams [4]. The former provide a hierarchical and graphical representation of the "flow" of data through a system inspired by the idea of *data flow graph*. They show the flow of data from external entities into the system, how these data are moved from one computation to another, and how they are logically stored. Similarly, *object flow diagrams* show the relationships among input objects, methods, and output objects in object-based models. Control flow diagrams represent the paths that can be traversed while executing a program. Each node in the graph represents a basic block, be it a single line or an entire function, and edges render how the execution jumps among them.

Moving to the second group, activity diagrams are similar to state diagrams [8], where the evolution of a system is rendered by the identification of the states, which characterize the element's life cycle, and of the transitions between them. A state transition can be constrained by the occurrence of an event and by an additional condition; its firing can cause the execution of an associated action. Mealy et al. propose different variations: Mealy assumes that actions be only associated with transitions, Moore only considers actions associated with states, and Harel's *state charts* [7] merge the two approaches with actions on both states and transitions, and enhance their flat model with nested and concurrent states.

The dynamic semantics of activity diagrams is clearly inspired by Petri Nets [9], which are a simple graphical formalism to specify the behavior of concurrent and parallel systems. The nodes are partitioned into places and transitions, with arcs that can only connect nodes of different type. Places may contain any number of tokens and a distribution of tokens over the places of a net is called a marking. A transition can only fire when there is at least a token in all its input places (i.e., those places connected to the transition by means of incoming edges), and its firing removes a token for all these places and produces a new one in each output place (i.e., a place connected to the transition through an outgoing edge). P/T nets only consider tokens as placeholders, while colored nets augment them with typed data and thus with firing conditions that become more articulated and can predicate on the tokens' values in the input places.

Activity diagrams also borrow from SDL (Specification and Description Language, [5]) as event handling. This is a specification language for the unambiguous description of the behavior of reactive and distributed systems. Originally, the notation was conceived for the specification of telecommunication systems, but currently its application is wider and includes process control and real-time applications in general. A system is specified as a set of interconnected abstract machines, which are extensions of finite state machines. SDL offers both a graphical and a textual representation and its last version (known as SDL-2000) is completely object-orientated.

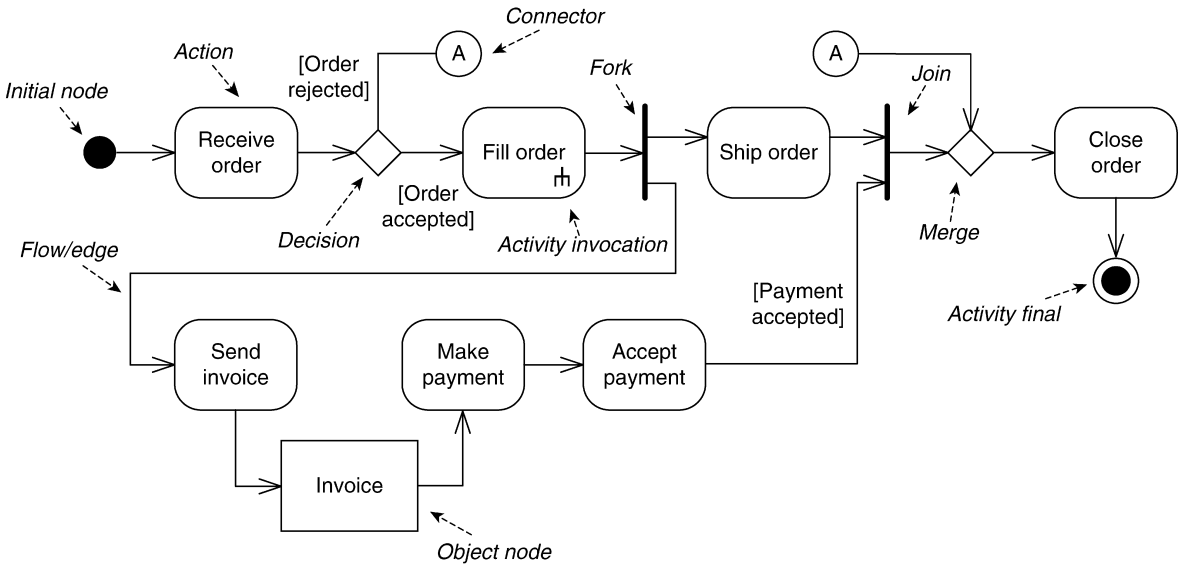
Foundations

Figure 1 addresses the well-known problem of order management and proposes a first *activity diagram* whose aim is twofold: it presents a possible formalization of the process, and it also introduces many of the concepts supplied by these diagrams.

Each atomic step is called *action*, with an *initial node* and *activity final* nodes to delimit their ordering as sequences, parallel threads, or conditional flows. A *fork* splits a single execution thread into a set of parallel ones, while a *join*, along with an optional *join specification* to constrain the unification, is used to re-synchronize the different threads into a single execution. Similarly, a *decision* creates alternative paths, and a *merge* re-unifies them. To avoid misunderstandings, each path must be decorated with the condition, in brackets, that must be verified to make the execution take that path.

The diagram of Fig. 1 also exemplifies the use of *connectors* to render *flows/edges* that might tangle the representation. This is nothing but an example, but the solution is interesting to avoid drawing flows that cross other elements or move all around the diagram. Another key feature is the use of a rake to indicate that action `Fill Order` is actually an *activity invocation*, and hides a hierarchical decomposition of actions into activities.

Besides the control flow, activity diagrams can also show the data/object flow among the actions. The use of *object nodes* allows users to state the artifacts exchanged between two actions, even if they are not directly connected by an edge. In many cases



Activity Diagrams. Figure 1. Example activity diagram.

control and object flows coincide, but this is not mandatory.

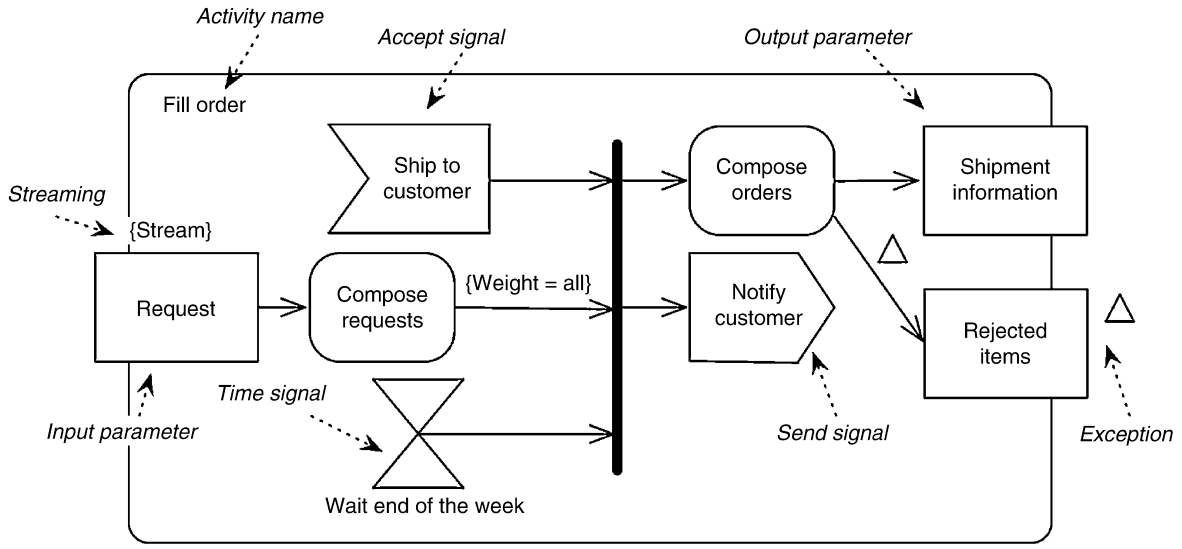
Activities can also comprise *input* and *output parameters* to render the idea that the activity's execution initiates when the inputs are available, and produces some outputs. For example, activity `Fill Order` of Fig. 2, which can be seen as a refinement of the invocation in Fig. 1, requires that at least one `Request` be present, but then it considers the parameter as a *stream*, and produces `Shipment Information` and `Rejected Items`. While the first outcome is the "normal" one, the second object is produced only in case of *exceptions* (rendered with a small triangle on both the object and the flow that produces it). In a stream, the flow is annotated from action `Compose Requests` to the join with its *weight* to mean that the subsequent processing must consider *all* the requests received when the composition starts.

The execution can also consider signals as enablers or outcomes of special-purpose actions. For example, Fig. 2 shows the use of an *accept signal*, to force that the composition of orders (`Compose Orders`) must be initiated by an external event, a *time signal*, to make the execution wait for a given timeframe (be it absolute or relative), and a *send signal*, to produce a notification to the customer as soon as the action starts.

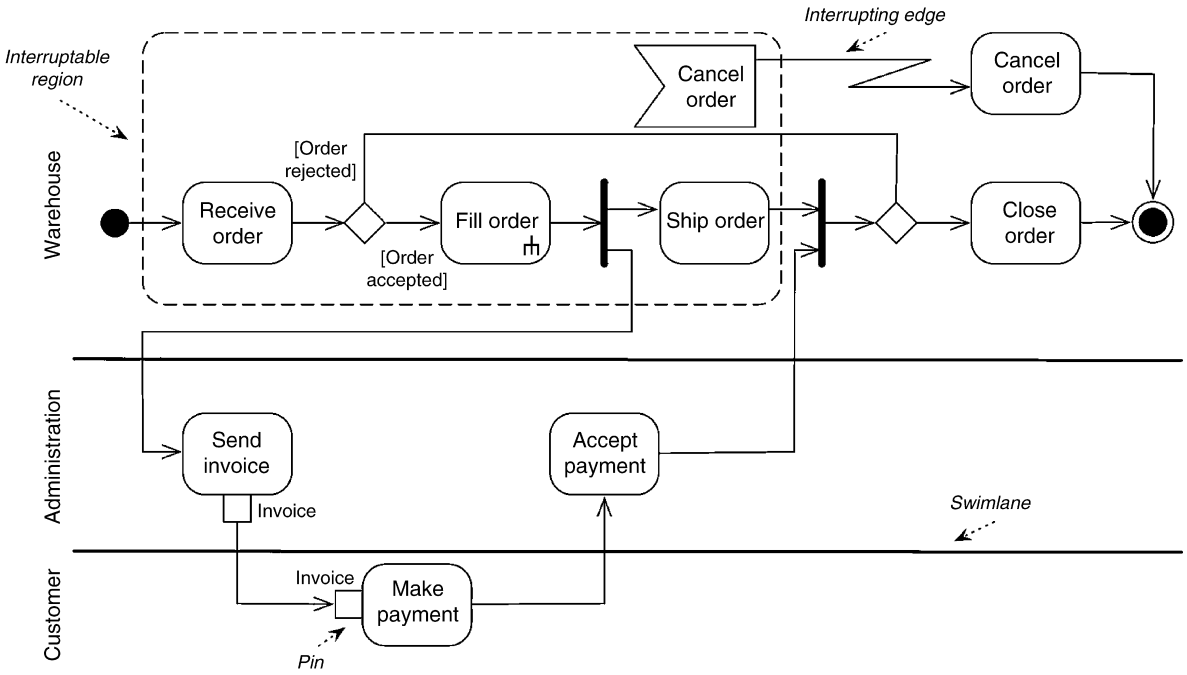
Basic diagrams can also be enriched with *swimlanes* to partition the different actions with respect to their responsibilities. Figure 3 shows a simple example: The primitive actions are the same as those of Fig. 1, but now they are associated with the three players in charge of activate the behaviors in the activity. The standard also supports hierarchical and multi-dimensional partitioning, that is, hierarchies of responsible actors or matrix-based partitions.

The `Warehouse` can also receive `Cancel Order` notifications to asynchronously interrupt the execution as soon as the external event arrivers. This is obtained by declaring an *interruptable region*, which contains the accept signal node and generates the interrupt that stops the computation in that region and moves the execution directly to action `Cancel Order` by means of an *interrupting edge*. More generally, this is a way to enrich diagrams with specialized *exception handlers* similarly to many modern programming and workflow languages. The figure also introduces *pins* as a compact way to render the objects exchanged between actions: empty boxes correspond to discrete elements, while filled ones refer to streams.

The discussion thus far considers the case in which the outcome of an action triggers a single execution of another action, but in some cases conditions may exist in which the "token" is structured and a single result triggers multiple executions of the same action. For example, if the example of Fig. 1 were slightly modified and after receiving an order, the user wants to check the items in it, a single execution of action



Activity Diagrams. Figure 2. Activity diagrams with signals.

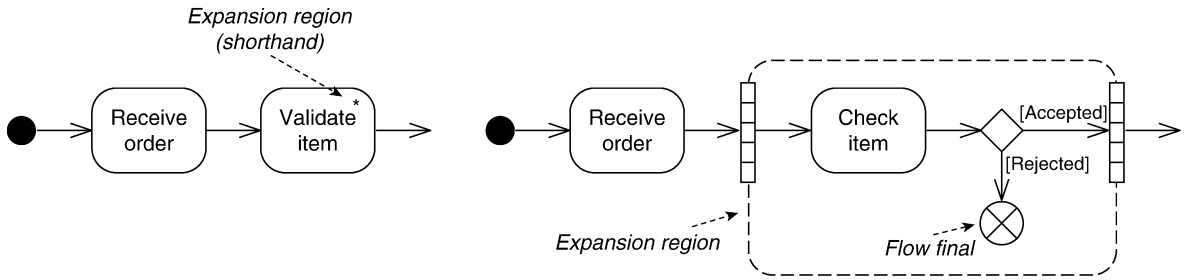


Activity Diagrams. Figure 3. Example swimlanes.

`Receive Order` would trigger multiple executions of `Validate Item`. This situation is depicted in the left-hand side of Fig. 4, where the star * conceives the information described so far.

The same problem can be addressed in a more complete way (right-hand side of figure) with an *expansion region*. The two arrays are supposed to store the input and output elements. In some cases, the number of input and output tokens is the same, but it might also be the case that the behavior in the region filters the incoming elements.

In the left-hand side of Fig. 4, it is assumed that some items are accepted and fill the output array, while others are rejected and thus their execution flow ends



Activity Diagrams. Figure 4. Expansion region.

there. This situation requires that a *flow final* be used to state that only the flow is ended and not the whole activity. Flow final nodes are a means to interrupt particular flows in this kind of regions, but also in loops or other similar cases.

The execution leaves an expansion region as soon as all the output tokens are available, that is, as soon as all the executions of the behavior embedded in the region are over. Notice that these executions can be carried out both concurrently (by annotating the rectangle with stereotype *concurrent*) or iteratively (with stereotype *iterative*). The next action considers the whole set of tokens as a single entity.

Further details about exceptions and other advanced elements, like pre- and post-conditions associated with single actions or whole activities, central buffers, and data stores are not discussed here, but the reader is referred to [11] for a thorough presentation.

Key Applications

Activity diagrams are usually employed to describe complex behaviors. This means that they are useful to model tangled processes, describe the actions that need to take place and when they should occur in use cases, render complicated algorithms, and model applications with parallel and alternative flows. Nowadays, these necessities belong to ICT specialists, like software engineering, requirements experts, and information systems architects, but also to experts in other fields (e.g., business analysts or production engineers) that need this kind of graphical notations to describe their solutions.

Activity diagrams can be used in isolation, when the user needs a pure control (data) flow notation, but they can also be adopted in conjunction with other modeling techniques such as interaction diagrams, state diagrams, or other UML diagrams. However, activity diagrams should not take the place of other diagrams. For example, even if the border between activity and state diagrams is sometimes blurred, activity diagrams provide a procedural decomposition of the problem under analysis, while state diagrams mostly concentrate on how studied elements behave. Moreover, activity diagrams do not give details about how objects behave or how they collaborate.

Cross-references

- ▶ Unified Modeling Language
- ▶ Web Services
- ▶ Workflow modeling

Recommended Reading

1. Arlow J. and Neustadt I. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design, 3rd edn. Addison-Wesley, Reading, MA, 2005.
2. Booch G., Rumbaugh J., and Jacobson I. The Unified Modeling Language User Guide, 2nd edn. Addison-Wesley, Reading, MA, 2005.
3. Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language, 3rd edn. Addison-Wesley, Reading, MA, 2003.
4. Gane C. and Sarson T. Structured System Analysis. Prentice-Hall, Englewood Cliffs, NJ, 1979.
5. Gaudin E., Najm E., and Reed R. In Proceedings of SDL 2007: Design for Dependable Systems, 13th International SDL Forum, LNCS, vol. 4745, Springer, 2007.
6. Goldstine H. The Computer from Pascal to Von Neumann. Princeton University Press, Princeton, NJ, 1972, pp. 266–267.
7. Harel D. and Naamad A. The STATEMATE Semantics of Statecharts. ACM Trans. Softw. Eng. Methodol., 5(4):293–333, 1996.
8. Hopcroft J. and Ullman J. Introduction to Automata Theory, Languages, and Computation. Addison-Wesley, Reading, MA, 2002.
9. Murata T. Petri Nets: Properties, Analysis, and Applications. Proc. IEEE, 77(4):541–580, 1989.
10. Object Management Group, http://www.omg.org/
11. OMG, Unified Modeling Language, http://www.uml.org/

Actors/Agents/Roles

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Workflow participant; Player; End user; Work performer

Definition

A resource that performs the work represented by a workflow activity instance.

This work is normally manifested as one or more work items assigned to the workflow participant via the worklist.

Key Points

These terms are normally applied to a human resource but it could conceptually include machine-based resources such as an intelligent agent.

Where an activity requires no human resource and is handled automatically by a computer application, the normal terminology for the machine-based resource is Invoked Application.

An Actor, Agent or Role may be identified directly within the business process definition, or (more normally) is identified by reference within the process definition to a role, which can then be filled by one or more of the resources available to the workflow system to operate in that role during process enactment.

Cross-references

► Activity
► Workflow Model

Ad hoc Retrieval models

► Information Retrieval Models

Adaptation

► Mediation

Adaptive Database Replication

► Autonomous Replication

Adaptive Interfaces

Maristella Matera
Politecnico di Milano University, Milan, Italy

Synonyms

Context-aware interfaces; Personalized interfaces

Definition

A specific class of user interfaces that are able to change in some way in response to different characteristics of the user, of the usage environment or of the task the user is supposed to accomplish. The aim is to improve the user's experience, by providing both interaction mechanisms and contents that best suit the specific situation of use.

Key Points

There are a number of ways in which interface adaptivity can be exploited to support user interaction. The interaction dimensions that are adapted vary among functionality (e.g., error correction or active help), presentation (user presentation of input to the system, system presentation of information to the user), and user tasks (e.g., task simplification based on the user's capabilities). Adaptivity along such dimensions is achieved by capturing and representing into some models a number of characteristics: the user's characteristics (preferences, experience, etc.); the tasks that the user accomplishes through the system; the characteristics of the information with which the user must be provided.

Due to current advances in communication and network technologies, adaptivity is now gaining momentum. Different types of mobile devices indeed offer support to access – at any time, from anywhere, and with any media – services and contents customized to the users' preferences and usage environments. In this new context, *content personalization*, based on user profile, has demonstrated its benefits for both users and content providers and has been

commonly recognized as fundamental factor for augmenting the overall effectiveness of applications. Going one step further, the new challenge in adaptive interfaces is now *context-awareness*. It can be interpreted as a natural evolution of personalization, addressing not only the user's identity and preferences, but also the environment that hosts users, applications, and their interaction, i.e., the *context*. Context-awareness, hence, aims at enhancing the application usefulness by taking into account a wide range of properties of the context of use.

Cross-references

► Visual Interaction

Adaptive Message-Oriented Middleware

► Adaptive Middleware for Message Queuing Systems

Adaptive Metric Techniques

► Learning Distance Measures

Adaptive Middleware for Message Queuing Systems

Christophe Taton[1], Noel De Palma[1], Sara Bouchenak[2]
[1]INPG - INRIA, Grenoble, France
[2]University of Grenoble I - INRIA, Grenoble, France

Synonyms

Autonomous message queuing systems; Adaptive message-oriented middleware; Autonomous message-oriented middleware

Definition

Distributed database systems are usually built on top of middleware solutions, such as message queuing systems. Adaptive message queuing systems are able to improve the performance of such a middleware through load balancing and queue provisioning.

Historical Background

The use of message oriented middlewares (MOMs) in the context of the Internet has evidenced a need for highly scalable and highly available MOM. A very promising approach to the above issue is to implement performance management as an autonomic software. The main advantages of this approach are: (i) Providing a high-level support for deploying and configuring applications reduces errors and administrator's efforts. (ii) Autonomic management allows the required reconfigurations to be performed without human intervention, thus improving the system reactivity and saving administrator's time. (iii) Autonomic management is a means to save hardware resources, as resources can be allocated only when required (dynamically upon failure or load peak) instead of pre-allocated.

Several parameters may impact the performance of MOMs. Self-optimization makes use of these parameters to improve the performance of the MOM. The proposed self-optimization approach is based on a queue clustering solution: a *clustered queue* is a set of queues each running on different servers and sharing clients. Self-optimization takes place in two parts: (i) the optimization of the clustered queue load-balancing and (ii) the dynamic provisioning of a queue in the clustered queue. The first part allows the overall improvement of the clustered queue performance while the second part optimizes the resource usage inside the clustered queue. Thus the idea is to create an autonomic system that fairly distributes client connections among the queues belonging to the clustered queue and dynamically adds and removes queues in the clustered queue depending on the load. This would allow to use the adequate number of queues at any time.

Foundations

Clustered Queues

A queue is a staging area that contains messages which have been sent by message producers and are waiting to be read by message consumers. A message is removed from the queue once it has been read. For scalability purpose, a queue can be replicated forming a clustered queue. The clustered queue feature provides a load balancing mechanism. A clustered queue is a cluster of queues (a given number of queue destinations knowing each other) that are able to exchange

messages depending on their load. Each queue of a cluster periodically reevaluates its load factor and sends the result to the other queues of the cluster. When a queue hosts more messages than it is authorized to do, and according to the load factors of the cluster, it distributes the extra messages to the other queues. When a queue is requested to deliver messages but is empty, it requests messages from the other queues of the cluster. This mechanism guarantees that no queue is hyper-active while some others are lazy, and tends to distribute the work load among the servers involved in the cluster.

Clustered Queue Performance

Clustered queues are standard queues that share a common pool of message producers and consumers, and that can exchange message to balance the load. All the queues of a clustered queue are supposed to be directly connected to each other. This allows message exchanges between the queues of a cluster in order to empty flooded queues and to fill draining queues.

The clustered queue Q_c is connected to N_c message producers and to M_c message consumers. Q_c is composed of standard queues $Q_i (i \in [1..k])$. Each queue Q_i is in charge of a subset of N_i message producers and of a subset of M_i message consumers:

$$\begin{cases} N_c = \sum_i N_i \\ M_c = \sum_i M_i \end{cases}$$

The distribution of the clients between the queues Q_i is described as follows: x_i (resp. y_i) is the fraction of message producers (resp. consumers) that are directed to Q_i.

$$\begin{cases} N_i = x_i \cdot N_c \\ M_i = y_i \cdot M_c \end{cases}, \begin{cases} \sum_i x_i = 1 \\ \sum_i y_i = 1 \end{cases}$$

The standard queue Q_i to which a consumer or producer is directed to cannot be changed after the client connection to the clustered queue. This way, the only action that may affect the client distribution among the queues is the selection of an adequate queue when the client connection is opened.

The clustered queue Q_c is characterized by its aggregate message production rate p_c and its aggregate message consumption rate c_c. The clustered queue Q_c also has a virtual clustered queue length l_c that aggregates the length of all contained standard queues:

$$l_c = \sum_i l_i = p_c - c_c, \begin{cases} p_c = \sum_i p_i \\ c_c = \sum_i c_i \end{cases}$$

The clustered queue length l_c obeys to the same law as a standard queue:

1. Q_c is globally stable when $\Delta l_c = 0$. This configuration ensures that the clustered queue is globally stable. However Q_c may observe local unstabilities if one of its queues is draining or is flooded.
2. If $\Delta l_c > 0$, the clustered queue will grow and eventually saturate; then message producers will have to wait.
3. If $\Delta l_c < 0$, the clustered queue will shrink until it is empty; then message consumers will also have to wait.

Now, considering that the clustered queue is globally stable, several scenarios that illustrate the impact of client distribution on performance are given below.

Optimal client distribution of the clustered queue Q_c is achieved when clients are fairly distributed among the k queues Q_i. Assuming that all queues and hosts have equivalent processing capabilities and that all producers (*resp.* consumers) have equivalent message production (*resp.* consumption) rates (and that all produced messages are equivalent: message cost is uniformly distributed), this means that:

$$\begin{cases} x_i = 1/k \\ y_i = 1/k \end{cases}, \begin{cases} N_i = \frac{N_c}{k} \\ M_i = \frac{M_c}{k} \end{cases}$$

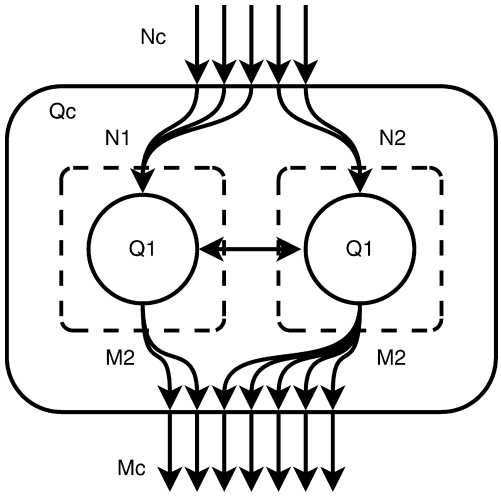
In these conditions, all queues Q_i are stable and the queue cluster is balanced. As a consequence, there are no internal queue-to-queue message exchanges, and performance is optimal. Queue clustering then provides a quasi-linear speedup.

The worst clients distribution appears when one queue only has message producers or only has message consumers. In the example depicted in Fig. 1, this is realized when:

$$\begin{cases} x_1 = 1 \\ y_1 = 0 \end{cases}, \begin{cases} x_2 = 0 \\ y_2 = 1 \end{cases}, \begin{cases} N_1 = N_c \\ M_1 = 0 \end{cases}, \begin{cases} N_2 = 0 \\ M_2 = M_c \end{cases}$$

Indeed, this configuration implies that the whole message production is directed to queue Q_1. Q_1 then forwards all messages to Q_2 that in turn delivers messages to the message consumers.

Local instability is observed when some queues Q_i of Q_c are unbalanced. This is characterized by a



Adaptive Middleware for Message Queuing Systems.
Figure 1. Clustered queue Q_c.

mismatch between the fraction of producers and the fraction of consumers directed to Q_i:

$$x_i \neq y_i$$

In the example showed in Fig. 1, Q_c is composed of two standard queues Q_1 and Q_2. A scenario of local instability can be envisioned with the following clients distribution:

$$\begin{cases} x_1 = 2/3 \\ y_1 = 1/3 \end{cases}, \begin{cases} x_2 = 1/3 \\ y_2 = 2/3 \end{cases}$$

This distribution implies that Q_1 is flooding and will have to enqueue messages, while Q_2 is draining and will see its consumer clients wait. However the queue cluster Q_c ensures the global stability of the system thanks to internal message exchanges from Q_1 to Q_2.

A stable and unfair distribution can be observed when the clustered queue is globally and locally stable, but the load is unfairly balanced within the queues. This happens when the client distribution is non-uniform.

In the example presented in Fig. 1, this can be realized by directing more clients to Q_1 than Q_2:

$$\begin{cases} x_1 = 2/3 \\ y_1 = 2/3 \end{cases}, \begin{cases} x_2 = 1/3 \\ y_2 = 1/3 \end{cases}$$

In this scenario, queue Q_1 processes two third of the load, while queue Q_2 only processes one third. Suc situation can lead to bad performance since Q_1 may saturates while Q_2 is lazy.

It is worthwhile to indicate that these scenarios may all happen since clients join and leave the system in an uncontrolled way. Indeed, the global stability of a (clustered) queue is under responsability of the application developper. For instance, the queue can be flooded for a period; it is assumed that it will get inverted and draining after, thus providing global stability over time.

Provisioning

The previous scenario of stable and non-optimal distribution raises the question of the capacity of a queue. The capacity C_i of standard queue Q_i is expressed as an optimal number of clients. The queue load L_i is then expressed as the ratio between its current number of clients and its capacity:

$$L_i = \frac{N_i + M_i}{C_i}$$

1. $L_i < 1$: queue Q_i is underloaded and thus lazy; the message throughput delivered by the queue can be improved and resources are wasted.
2. $L_i > 1$: queue Q_i is overloaded and may saturate; this induces a decreased message throughput and eventually leads to thrashing.
3. $L_i = 1$: queue Q_i is fairly loaded and delivers its optimal message throughput.

These parameters and indicators are transposed to queue clusters. The clustered queue Q_c is characterized by its aggregated capacity C_c and its global load L_c:

$$C_c = \sum_i C_i, \; L_c = \frac{N_c + M_c}{C_c} = \frac{\sum_i L_i \cdot C_i}{\sum_i C_i}$$

The load of a clustered queue obeys to the same law as the load of a standard queue.

However a clustered queue allows to control k, the number of inside standard queues, and thus to control its aggregated capacity $C_c = \sum_{i=1}^{k} C_i$. This control is indeed operated with a re-evaluation of the clustered queue provisioning.

1. When $L_c < 1$, the clustered queue is underloaded: if the clients distribution is optimal, then all the standard queues inside the cluster will be underloaded. However, as the client distribution may be non-optimal, some of the single queues may be overloaded, even if the cluster is globally lazy. If the load is too low, then some queues may be removed from the cluster.

2. When $L_c > 1$, the clustered queue is overloaded: even if the distribution of clients over the queues is optimal, there will exist at least one standard queue that will be overloaded. One way to handle this case is to re-provision the clustered queue by inserting one or more queues into the cluster.

Control Rules for a Self-Optimizing Clustered Queue

The global clients distribution D of the clustered queue Q_c is captured by the fractions of message producers x_i and consumers y_i. The optimal clients distribution D_{opt} is realized when all queues are stable ($\forall i\ x_i = y_i$) and when the load is fairly balanced over all queues ($\forall i, j x_i = x_j, y_i = y_j$). This implies that the optimal distribution is reached when $x_i = y_i = 1/k$.

$$D = \begin{bmatrix} x_1 & y_1 \\ \vdots & \vdots \\ x_k & y_k \end{bmatrix}, \quad D_{opt} = \begin{bmatrix} 1/k & 1/k \\ \vdots & \vdots \\ 1/k & 1/k \end{bmatrix}$$

Local instabilities are characterized by a mismatch between the fraction of message producers x_i and consumers y_i on a standard queue. The purpose of this rule is the stability of all standard queues so as to minimize internal queue-to-queue message transfer.

1. $[(R_1)]$ $x_i > y_i$: Q_i is flooding with more message production than consumption and should then seek more consumers and/or fewer producers.
2. $[(R_2)]$ $x_i < y_i$: Q_i is draining with more message consumption than production and should then seek more producers and/or fewer consumers.

Load balancing rules control the load applied to a single standard queue. The goal is then to enforce a fair load balancing over all queues.

1. $[(R_3)]$ $L_i > 1$: Q_i is overloaded and should avoid accepting new clients as it may degrade its performance.
2. $[(R_4)]$ $L_i < 1$: Q_i is underloaded and should request more clients so as to optimize resource usage.

Global provisioning rules control the load applied to the whole clustered queue. These rules target the optimal size of the clustered queue while the load applied to the system evolves.

1. $[(R_5)]$ $L_c > 1$: the queue cluster is overloaded and requires an increased capacity to handle all its clients in an optimal way.
2. $[(R_6)]$ $L_c < 1$: the queue cluster is underloaded and could accept a decrease in capacity.

Key Applications

Adaptive middleware for message queuing systems helps building autonomous distributed systems to improve their performance while minimizing their resource usage, such as distributed Internet services and distributed information systems.

Cross-references

- ▶ Distributed Database Systems
- ▶ Distributed DBMS
- ▶ Message Queuing Systems

Recommended Reading

1. Aron M., Druschel P., and Zwaenepoel W. Cluster reserves: a mechanism for resource management in cluster-based network servers. In Proc. 2000 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comp. Syst., 2000, pp. 90–101.
2. Menth M. and Henjes R. Analysis of the message waiting time for the fioranoMQ JMS server. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2006, pp. 1.
3. Shen K., Tang H., Yang T., and Chu L. Integrated resource management for cluster-based internet services. In Proc. 5th USENIX Symp. on Operating System Design and Implementation, 2002.
4. Urgaonkar B. and Shenoy P. Sharc: Managing CPU and network bandwidth in shared clusters. IEEE Trans. Parall. Distrib. Syst., 15(1):2–17, 2004.
5. Zhu H., Ti H., and Yang Y. Demand-driven service differentiation in cluster-based network servers. In Proc. 20th Annual Joint Conf. of the IEEE Computer and Communications Societies, vol. 2, 2001, pp. 679–688.

Adaptive Query Optimization

▶ Adaptive Query Processing

Adaptive Query Processing

Evaggelia Pitoura
University of Ioannina, Ioannina, Greece

Synonyms

Adaptive query optimization; Eddies; Autonomic query processing

Definition

While in traditional query processing, a query is first optimized and then executed, *adaptive query processing* techniques use runtime feedback to modify query processing in a way that provides better response time, more efficient CPU utilization or more useful incremental results. Adaptive query processing makes query processing more robust to optimizer mistakes, unknown statistics, and dynamically changing data, runtime and workload characteristics. The spectrum of adaptive query processing techniques is quite broad: they may span the executions of multiple queries or adapt within the execution of a single query; they may affect the query plan being executed or just the scheduling of operations within the plan.

Key Points

Conventional query processing follows an optimize-then-execute strategy: after generating alternative query plans, the query optimizer selects the most cost-efficient among them and passes it to the execution engine that directly executes it, typically with little or no runtime decision-making. As queries become more complex, this strategy faces many limitations such as missing statistics, unexpected correlations, and dynamically changing data, runtime, and workload characteristics. These problems are aggregated in the case of long-running queries over data streams as well as in the case of queries over multiple potentially heterogeneous data sources across wide-area networks. Adaptive query processing tries to address these shortcomings by using feedback during query execution to tune query processing. The goal is to increase throughput, improve response time or provide more useful incremental results.

To implement adaptivity, regular query execution is supplemented with a control system for monitoring and analyzing at run-time various parameters that affect query execution. Based on this analysis, certain decisions are made about how the system behavior should be changed. Clearly, this may introduce considerable overheads.

The complete space of adaptive query processing techniques is quite broad and varied. Adaptability may be applied to query execution of multiple queries or just a single one. It may also affect the whole query plan being executed or just the scheduling of operations within the plan. Adaptability techniques also differ on how much they interleave plan generation and execution. Some techniques interleave planning and execution just a few times, by just having the plan re-optimized at specific points, whereas other techniques interleave planning and execution to the point where they are not even clearly distinguishable.

A number of fundamental adaptability techniques include:

- *Horizontal partitioning*, where different plans are used on different portions of the data. Partitioning may be explicit or implicit in the functioning of the operator.
- *Query execution by tuple routing*, where query execution is treated as the process of routing tuples through operators and adaptability is achieved by changing the order in which tuples are routed.
- *Plan partitioning*, where execution progresses in stages, by interleaving optimization and execution steps at a number of well-defined points during query execution.
- *Runtime binding decisions*, where certain plan choices are deferred until runtime, allowing the execution engine to select among several alternative plans by potentially re-invoking the optimizer.
- *In-operator adaptive logic*, where scheduling and other decisions are made part of the individual query operators, rather than the optimizer.

Many adaptability techniques rely on a symmetric hash join operator that offers a non-blocking variant of join by building hash tables on both the input relations. When an input tuple is read, it is stored in the appropriate hash table and probed against the opposite table, thus producing incremental output. The symmetric hash join operator can process data from either input, depending on availability. It also enables additional adaptivity, since it has frequent moments of symmetry, that is, points at which the join order can be changed without compromising correctness or losing work.

The eddy operator provides an example of fine-grained run-time control by tuple routing through operators. An eddy is used as a tuple router; it monitors execution, and makes routing decisions for the tuples. Eddies achieve adaptability by simply changing the order in which the tuples are routed through the operators. The degree of adaptability achieved depends on the type of the operators. Pipelined operators, such as the symmetric hash join, offer the most freedom, whereas, blocking operators, such as the sort-merge

join, are less suitable since they do not produce output before consuming the input relations in their entirety.

Cross-references

- ▶ Adaptive Stream Processing
- ▶ Cost Estimation
- ▶ Multi-query Optimization
- ▶ Query Optimization
- ▶ Query Processing

Recommended Reading

1. Avnur R. and Hellerstein J.M. Eddies: continuously adaptive query processing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 261–272.
2. Babu S. and Bizarro P. Adaptive query processing in the looking glass. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 238–249.
3. Deshpande A., Ives Z.G., and Raman V. Adaptive query processing. Found. Trends Databases, 1(1):1–140, 2007.

Adaptive Stream Processing

Zachary Ives
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Adaptive query processing

Definition

When querying long-lived data streams, the characteristics of the data may change over time or data may arrive in bursts – hence, the traditional model of optimizing a query prior to executing it is insufficient. As a result, most data stream management systems employ feedback-driven *adaptive stream processing*, which continuously re-optimizes the query execution plan based on data and stream properties, in order to meet certain performance or resource consumption goals. Adaptive stream processing is a special case of the more general problem of *adaptive query processing*, with the special property that intermediate results are bounded in size (by stream windows), but where query processing may have quality-of-service constraints.

Historical Background

The field of adaptive stream processing emerged in the early 2000s, as two separate developments converged. *Adaptive* techniques for database query processing had become an area of increasing interest as Web and integration applications exceeded the capabilities of conventional static query processing [10]. Simultaneously, a number of data stream management systems [1,6,8,12] were emerging, and each of these needed capabilities for query optimization. This led to a common approach of developing feedback-based re-optimization strategies for stream query computation. In contrast to Web-based adaptive query processing techniques, the focus in adaptive stream processing has especially been on maintaining quality of service under overload conditions.

Foundations

Data stream management systems (DSMSs) typically face two challenges in query processing. First, the data to be processed comes from remote feeds that may be subject to significant variations in distribution or arrival rates over the lifetime of the query, meaning that no single query evaluation strategy may be appropriate over the entirety of execution. Second, DSMSs may be *underprovisioned* in terms of their ability to handle bursty input at its maximum rate, and yet may still need to meet certain *quality-of-service* or resource constraints (e.g., they may need to ensure data is processed within some latency bound). These two challenges have led to two classes of adaptive stream processing techniques: those that attempt to *minimize the cost* of computing query results from the input data (the problem traditionally faced by query optimization), and those that attempt to manage query processing, possibly at reduced accuracy, in the presence of *limited resources*. This article provides an overview of significant work in each area.

Minimizing Computation Cost

The problem of adaptive query processing to minimize computation cost has been well-studied in a variety of settings [10]. What makes the adaptive stream processing setting unique (and unusually tractable) is the fact that joins are performed over *sliding windows* with size bounds: As the data stream exceeds the window size, old data values are expired. This means intermediate state within a query plan operator has constant maximum size; as opposed to being bounded by the size of the input data. Thus a windowed join operator can be modeled as a pair of filter operators, each of which joins its input with the bounded intermediate state produced from the other input. Optimization of joins

in data stream management systems becomes a minor variation on the problem of optimizing selection or filtering operators; hence certain theoretical optimality guarantees can actually be made.

Eddies Eddies [2,11,14] are composite dataflow operators that model select-project-join expressions. An eddy consists of a *tuple router*, plus a set of primitive *query operators* that run concurrently and each have input queues. Eddies come in several variations; the one proposed for distributed stream management uses *state modules* (SteMs) [14,11]. Figure 1 shows an example of such an eddy for a simplified stream SQL query, which joins three streams and applies a selection predicate over them.

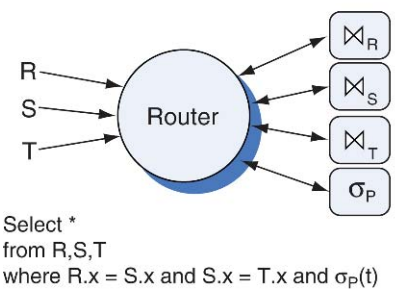
Eddy creation. The eddy is created prior to execution by an optimizer: every selection operation (σ_P in the example) is converted to a corresponding operator; additionally, each base relation to be joined is given a *state module*, keyed on the join attribute, to hold the intermediate state for each base relation [14] ($\bowtie_R$, $\bowtie_S$, $\bowtie_T$). If a base relation appears with multiple different join attributes, then it may require multiple SteMs. In general, the state module can be thought of as one of the hash tables within a symmetric or pipelined hash join. The optimizer also determines whether the semantics of the query force certain operators to execute before others. Such constraints are expressed in an internal routing table, illustrated on the right side of the figure. As a tuple is processed, it is annotated with a *tuple signature* specifying what input streams' data it contains and what operator may have last modified it. The routing table is a map from the tuple signature to a set of *valid routing destinations*, those operators that can successfully process a tuple with that particular signature.

Query execution/tuple routing. Initially, a tuple from an input data stream (R, S, or T) flows into the eddy router. The eddy (i) adds the data to the associated SteM or SteMs, and (ii) consults the routing table to determine the set of possible destination operators. It then chooses a destination (using a *policy* to be described later) and sends the tuple to the operator. The operator then either *filters* the tuple, or *produces* one or more output tuples, as a result of applying selection conditions or joining with the data within a SteM. Output tuples are marked as having been processed by the operator that produced them. If they have been processed by all operators, they will be sent to the query output, and if not, they will be sent back to the eddy's router and to one of the remaining operators.

Routing policies. The problem of choosing among alternate routing destinations has been addressed with a variety of strategies.

Tickets and lottery scheduling [2]. In this scheme, each operator receives a *ticket* for each tuple it receives from the router, and it returns the ticket each time it outputs a tuple to the router. Over time, each operator is expected to have a number of tickets proportional to $(1 - p)$ where p is the operator's selectivity. The router holds a *lottery* among valid routing destinations, where each operator's chance of winning is proportional to its number of tickets. Additionally, as a flow control mechanism, each operator has an input queue, and if this queue fills, then the operator may not participate in the lottery.

Deterministic with batching [9]. A later scheme was developed to reduce the per-tuple overhead of eddies by choosing destinations for batches of tuples. Here, each operator's selectivity is explicitly monitored and each predicate is assumed to be independent. Periodically, a *rank ordering* algorithm is used to choose a



Tuple signature

Contains	*From*	*Valid routing destinations*
{R}	{}	{⋈S, ⋈T, σP}
{S}	{}	{⋈R, ⋈T, σP}
{T}	{}	{⋈R, ⋈S, σP}
{S,T}	{⋈S, ⋈T}	{⋈R, σP}
{R}	{σP}	{⋈S, ⋈T}
…	…	…

Adaptive Stream Processing. Figure 1. Illustration of eddy with SteMs.

destination for a *batch* of tuples: the rank ordering algorithm sorts predicates in decreasing order of $c_i/(1-p_i)$, where c_i is the cost of the applying predicate σ_i and p_i is its selectivity.

Content-based routing [7]. (CBR) attempts to learn correlations between attribute values and selectivities. Using sampling, the system determines for each operator the attribute most strongly correlated with its selectivity – this is termed the *classifier attribute*. CBR then builds a table characterizing all operators' selectivities for different values of each classifier attribute. Under this policy, when the eddy needs to route a tuple, it first looks up the tuple's classifier attribute values in the table and determines the destination operators' selectivities. It routes the tuple probabilistically, choosing a next operator with probability inversely proportional to its selectivity.

Other optimization strategies. An alternative strategy that does not use the eddies framework is the *adaptive greedy* [5] (A-greedy) algorithm. A-greedy continuously monitors the selectivities of query predicates using a *sliding window profile*, a table with one Boolean attribute for each predicate in the query, and sampling. As a tuple is processed by the query, it may be chosen for sampling into the sliding window profile – if so, it is tested against every query predicate. The vector of Boolean results is added as a row to the sliding window profile. Then the sliding window profile is then used to create a *matrix view* $V[i, j]$ containing, for each predicate σ_i, the number of tuples in the profile that satisfy $\sigma_1 \ldots \sigma_{i-1}$ but not σ_j. From this matrix view, the reoptimizer seeks to maintain the constraint that the ith operation over an input tuple must have the lowest cost/selectivity ratio $c_i/(1-p(S_i|S_1,\ldots,S_{i-1}))$. The overall strategy has one of the few performance guarantees in the adaptive query processing space: if data properties were to converge, then performance would be within a factor of 4 of optimal [5].

Managing Resource Consumption

A common challenge in data stream management systems is limiting the use of resources – or accommodating limited resources while maintaining quality of service, in the case of bursty data. We discuss three different problems that have been studied: load shedding to ensure input data is processed by the CPU as fast as it arrives, minimizing buffering and memory consumption during data bursts, and minimizing network communication with remote streaming sites.

Load Shedding. Allows the system to selectively drop data items to ensure it can process data as it arrives. Both the Aurora and STREAM DSMSs focused heavily on adaptive load shedding.

Aurora. In the Aurora DSMS [15], load shedding for a variety of query types are supported: the main requirement is that the user has a *utility function* describing the value of output data relative to how much of it has been dropped. The system seeks to place load shedding operators in the query plan in a way that maximizes the user's utility function while the system achieves sufficient throughput. Aurora precomputes conditional load shedding plans, in the form of a *load shedding road map* (LRSM) containing a sequence of plans that shed progressively more load; this enables the runtime system to rapidly move to strategies that shed more or less load.

LRSMs are created using the following heuristics: first, load shedding points are only inserted at data input points or at points in which data is split to two or more operators. Second, for each load shedding point, a *loss/gain ratio* is computed: this is the reduction in output utility divided by the gain in cycles, $R(p \cdot L - D)$, where R is the input rate into the drop point, p is the ratio of tuples to be dropped, L is the amount of system load flowing from the drop point, and D is the cost of the drop operator. Drop operators are injected at load shedding points in decreasing order of loss/gain ratio. Two different types of drops are considered using the same framework: *random drop*, in which an operator is placed in the query plan to randomly drop some fraction p of tuples; and *semantic drop*, which drops the p tuples of lowest utility. Aurora assumes for the latter case that there exists a utility function describing the relative worth of different attribute values.

Stanford STREAM. The Stanford STREAM system [4] focuses on aggregate (particularly SUM) queries. Again the goal is to process data at the rate it arrives, while minimizing the inaccuracy in query answers: specifically, the goal is to minimize the *maximum relative error across all queries*, where the relative error of a query is the difference between actual and approximate value, divided by the actual value.

A Statistics Manager monitors computation and provides estimates of each operator's selectivity and its running time, as well as the mean value and standard deviation of each query q_i's aggregate operator. For each q_i, STREAM computes an error threshold C_i, based on the mean, standard deviation, and number of values. (The results are highly technical so the reader is referred to [4] for more details.) A sampling rate P_i is chosen for query q_i that satisfies $P_i \geq C_i/\in_i$, where $\in_i$ is the allowable relative error for the query.

As in Aurora's load shedding scheme, STREAM only inserts load shedding operators at the inputs or at the start of shared segments. Moreover, if a node has a set of children who all need to shed load, then a portion of the load shedding can be "pulled up" to the parent node, and all other nodes can be set to shed some amount of additional load *relative* to this. Based on this observation, STREAM creates a query dataflow graph in which each path from source to sink initially traverses through a load shedding operator whose sampling rate is determined by the desired error rate, followed by additional load shedding operators whose sampling rate is expressed relative to that first operator. STREAM iterates over each path, determines a sampling rate for the initial load shedding operator to satisfy the load constraint, and then computes the maximum relative error for any query. From this, it can set the load shedding rates for individual operators.

Memory Minimization. STREAM also addresses the problem of minimizing the amount of space required to buffer data in the presence of burstiness [3]. The Chain algorithm begins by defining a *progress chart* for each operator in the query plan: this chart plots the relative size of the operator output versus the time it takes to compute. A point is plotted at time 0 with the full size of the input, representing the start of the query; then each operator is given a point according to its cost and relative output size. Now a *lower envelope* is plotted on the progress chart: starting with the initial point at time 0, the *steepest* line is plotted to any operator to the right of this point; from the point at the end of the first line, the next steepest line is plotted to a successor operator; etc. Each line segment (and the operators whose points are plotted beside it) represents a chain, and operators within a chain are scheduled together. During query processing, at each time "tick," the scheduler considers all tuples that have been output by any chain. The tuple that lies on the segment with *steepest slope* is the one that is scheduled next; as a tie-breaker, the earliest such tuple is scheduled. This Chain algorithm is proven to be near-optimal (differing by at most one unit of memory per operator path for queries where selectivity is at most one).

Minimizing Communication. In some cases, the constrained resource is the network rather than CPU or memory. Olston et al. [13] develop a scheme for reducing network I/O for AVERAGE queries, by using accuracy bounds. Each remote object O is given a *bound width* w_O: the remote site will only notify the central query processor if O's value V falls outside this bound. Meanwhile, the central site maintains a *bound cache* with the last value and the bound width for every object.

If given a *precision constraint* δ_j for each query Q_j, then if the query processor is to provide query answers within δ_j, the sum of the bound widths for the data objects of Q_j must not exceed δ_j times the number of objects. The challenge lies in the selection of widths for the objects.

Periodically, the system tries to tighten all bounds, in case values have become more stable; objects whose values fall outside the new bounds get reported back to the central site. Now some of those objects' bounds must be loosened in a way that maintains the precision constraints over all queries. Each object O is given a *burden score* equal to $c_O/(p_O w_O)$, where c_O is the cost of sending the object, w_O is its bound width, and p_O is the frequency of updates since the previous width adjustment. Using an approximation method based on an iterative linear equation solver, Olston et al. compute a *burden target* for each query, i.e., the lowest overall burden score required to always meet the query's precision constraint. Next, each object is assigned a *deviation*, which is the maximum difference between the object's burden score and any query's burden target. Finally, a queried objects' bounds are adjusted in decreasing order of deviation, and each object's bound is increased by the largest amount that still conforms to the precision constraint for every query.

Key Applications

Data stream management systems have seen significant adoption in areas such as sensor monitoring and processing of financial information. When there are

associated quality-of-service constraints that might require load shedding, or when the properties of the data are subject to significant change, adaptive stream processing becomes vitally important.

Future Directions

One of the most promising directions of future study is how to best use a combination of offline modeling, selective probing (in parallel with normal query execution), and feedback from query execution to find optimal strategies quickly. Algorithms with certain optimality guarantees are being explored in the online learning and theory communities (e.g., the *k*-armed bandit problem), and such work may lead to new improvements in adaptive stream processing.

Cross-references

- ▶ Distributed Stream
- ▶ Query Processor
- ▶ Stream Processing

Recommended Reading

1. Abadi D.J., 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. Avnur R. and Hellerstein J.M. Eddies: continuously adaptive query processing. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 261–272.
3. 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, pp. 253–264.
4. Babcock B., Datar M., and Motwani R. Load shedding for aggregation queries over data streams. In Proc. 20th Int. Conf. on Data Engineering, 2004, p. 350.
5. Babu S., Motwani R., Munagala K., Nishizawa I., and Widom J. Adaptive ordering of pipelined stream filters. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 407–418.
6. Balazinska M., BalaKrishnan H., and Stonebraker M. Demonstration: load management and high availability in the Medusa distributed stream processing system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 929–930.
7. Bizarro P., Babu S., DeWitt D.J., and Widom J. Content-based routing: different plans for different data. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 757–768.
8. 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.
9. Deshpande A. An initial study of overheads of eddies. ACM SIGMOD Rec., 33(1):44–49, 2004.
10. Deshpande A., Ives Z., and Raman V. Adaptive query processing. Found. Trends Databases, 1(1):1–140, 2007.
11. Madden S., Shah M.A., Hellerstein J.M., and Raman V. Continuously adaptive continuous queries over streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 49–60.
12. Motwani R., Widom J., Arasu A., Babcock B., Babu S., Datar M., Manku G., Olston C., Rosenstein J., and Varma R. Query processing, resource management, and approximation in a data stream management system. In Proc. 1st Biennial Conf. on Innovative Data Systems Research, 2003.
13. Olston C.,Jiang J., and Widom J., Adaptive filters for continuous queries over distributed data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 563–574.
14. Raman V., Deshpande A., and Hellerstein J.M. Using state modules for adaptive query processing. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 353–366.
15. Tatbul N., Cetintemel U., Zdonik S.B., Cherniack M., and Stonebraker M. Load shedding in a data stream manager. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 309–320.

Adaptive Workflow/Process Management

- ▶ Workflow Evolution

ADBMS

- ▶ Active Database Management System Architecture

ADBMS Framework

- ▶ Active Database Management System Architecture

ADBMS Infrastructure

- ▶ Active Database Management System Architecture

Adding Noise

- ▶ Matrix Masking

Additive Noise

► Noise Addition

Administration Model for RBAC

Yue Zhang, James B. D. Joshi
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

ARBAC97; SARBAC

Definition

The central ideal of administration model for RBAC is to use the role itself to manage roles. There are two well-known families of administration RBAC models.

Administrative RBAC

The Administrative RBAC family of models known as ARBAC97 [3] introduces administrative roles that are used to manage the regular roles. These roles can form a role hierarchy and may have constraints. ARBAC97 consists of three administrative models, the user-role assignment (URA97) model, the permission-role assignment (PRA97) model, and the role-role administration (RRA97) model. URA97 defines which administrative roles can assign which users to which regular roles by means of the relation: *can_assign*. Similarly, PRA97 defines which administrative roles can assign which permissions to which regular roles by means of the relation: *can_assignp*. Each of these relations also has a counterpart for revoking the assignment (e.g., *can_revoke*). RRA97 defines which administrative roles can change the structure (add roles, delete roles, add edges, etc.) of which range of the regular roles using the notion of *encapsulated range* and the relation: *can_modify*.

Scoped Administrative RBAC

The SARBAC model uses the notion of *administrative scope* to ensure that any operations executed by a role *r* will not affect other roles due to the hierarchical relations among them [1]. There are no special administrative roles in SARBAC, and each regular role has a *scope* of other regular roles called *administrative scope* that can be managed by it. Each role can only be managed by its administrators. For example, a senior-most role should be able to manage all its junior roles.

Key Points

ARBAC model is the first known role-based administration model and uses the notion of *range* and *encapsulated range*. Role *range* is essentially a set of regular roles. To avoid undesirable side effects, RRA97 requires that all role ranges in the *can_modify* relation be encapsulated, which means the range should have exactly one senior-most role and one junior-most role. Sandhu et al. later extended the ARBAC97 model into ARBAC99 model where the notion of mobile and immobile user/permission was introduced [4]. Oh et al. later extended ARBAC99 to ARBAC02 by adding the notion of organizational structure to redefine the user-role assignment and the role-permission assignment [2]. Recently, Zhang et al. have proposed an ARBAC07 model that extends the family of ARBAC models to deal with an RBAC model that allows hybrid hierarchies to co-exit [6].

SARBAC

The most important notion in SARBAC is that of the *administrative scope*, which is similar to the notion of *encapsulated range* in ARBAC97. A role *r* is said to be within to be the administrative scope of another role *a* if every path upwards from *r* goes through *a*, and *a* is said to be the administrator of *r*. SARBAC also consists of three models: SARBAC-RHA, SARBAC-URA, and SARBAC-PRA. In SARBAC-RHA, each role can only administer the roles that are within its own administrative scope. The operations include adding roles, deleting roles, adding permissions, and deleting permissions. The semantics for SARBAC-URA and SARBAC-PRA is similar to URA97 and PRA97. The administrative scope can change dynamically. Zhang et al. have extended SARBAC to also deal with hybrid hierarchy [5].

Cross-references

► Role Based Access Control

Recommended Reading

1. Crampton J. and Loizou G. Administrative scope: a foundation for role-based administrative models. ACM Trans. Inf. Syst. Secur., 6(2):201–231, 2003.
2. Oh S. and Sandhu R. A model for role administration using organization structure. In Proc. 7th ACM Symp. on Access Control Models and Technologies, 2002, pp. 155–162.

3. Sandhu R., Bhamidipati V., and Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Trans. Inf. Syst. Secur., 2(1):105–135, 1999.
4. Sandhu R. and Munawer Q. The ARBAC99 model for administration of roles (1999). In Proc. 15th Computer Security Applications Conf. Arizona, 1999, pp. 229.
5. Zhang Y., James B., and Joshi D. "SARBAC07: scoped administration model for RBAC with hybrid hierarchy. In Proc. 3rd Int. Symp. on Information Assurance and Security, 2007, pp. 149–154.
6. Zhang Y. and Joshi J.B.D. ARBAC07: a role based administration model for RBAC with hybrid hierarchy. In Proc. IEEE Int. Conf. Information Reuse and Integration, 2007, pp. 196–202.

Administration Wizards

Philippe Bonnet[1], Dennis Shasha[2]
[1]University of Copenhagen, Copenhagen, Denmark
[2]New York University, New York, NY, USA

Definition

Modern database systems provide a collection of utilities and programs to assist a database administrator with tasks such as database installation and configuration, import/export, indexing (index wizards are covered in the self-management entry), and backup/restore.

Historical Background

Database Administrators have been skeptical of any form of automation as long as they could control the performance and security of a relatively straightforward installation. The advent of enterprise data management towards the end of the 1990s, where few administrators became responsible for many, possibly diverse database servers, has led to the use of graphical automation tools. In the mid-1990s, third party vendors introduced such tools. With SQL Server 6.5, Microsoft was the first constructor to provide an administration wizard.

Foundations

Installation and Configuration

Database servers are configured using hundreds of parameters that control everything buffer size, file layout, concurrency control options and so on. They are either set statically in a configuration file before the server is started, or dynamically while the server is running. Out-of-the-box database servers are equipped with a limited set of typical configurations.

The installation/configuration wizard is a graphical user interface that guides the administrator through the initial server configuration. The interface provides high-level choices (e.g., OLTP vs. OLAP workload), or simple questions (e.g., number of concurrent users) that are mapped onto database configuration values (log buffer size and thread pool size respectively).

Data Import/Export

Import/export wizards are graphical tools that help database administrators map a database schema with an external data format (e.g., XML, CSV, PDF), or generate scripts that automate the transfer of data between a database and an external data source (possibly another database server).

Back-up/Restore

Back-up/restore wizards automate the back-up procedure given a few input arguments: complete/incremental backup, scope of the back-up/restore operations (file, tablespace, database), target directory.

Key Applications

Automation of the central database administration tasks.

Cross-references

▶ Self-Management

Recommended Reading

1. Bersinic D. and Gile S. Portable DBA: SQL Server. McGraw Hill, New York, 2004.
2. Schumacher R. DBA Tools Today. DBMS Magazine, January 1997.

Advanced Transaction Models

▶ Extended Transaction Models and the ACTA Framework
▶ Generalization of ACID Properties
▶ Open Nested Transaction Models

Adversarial Information Retrieval

▶ Web Spam Detection

Affix Removal

▶ Stemming

AFI

▶ Approximation of Frequent Itemsets

Aggregate Queries in P2P Systems

▶ Approximate Queries in Peer-to-Peer Systems

Aggregation

▶ Abstraction

Aggregation Algorithms for Middleware Systems

▶ Top-k Selection Queries on Multimedia Datasets

Aggregation and Threshold Algorithms for XML

▶ Ranked XML Processing

Aggregation: Expressiveness and Containment

Sara Cohen
The Hebrew University of Jerusalem, Jerusalem, Israel

Definition

An *aggregate function* is a function that receives as input a multiset of values, and returns a single value. For example, the aggregate function `count` returns the number of input values. An *aggregate query* is simply a query that mentions an aggregate function, usually as part of its output. Aggregate queries are commonly used to retrieve concise information from a database, since they can cover many data items, while returning few. Aggregation is allowed in SQL, and the addition of aggregation to other query languages, such as relational algebra and datalog, has been studied.

The problem of determining *query expressiveness* is to characterize the types of queries that can be expressed in a given query language. The study of query expressiveness for languages with aggregation is often focused on determining how aggregation increases the ability to formulate queries. It has been shown that relational algebra with aggregation (which models SQL) has a *locality property*.

Query containment is the problem of determining, for any two given queries q and q', whether $q(D) \subseteq q'(D)$, for all databases D, where $q(D)$ is the result of applying q to D. Similarly, the *query equivalence problem* is to determine whether $q(D) = q'(D)$ for all databases D. For aggregate queries, it seems that characterizing query equivalence may be easier than characterizing query containment. In particular, almost all known results on query containment for aggregate queries are derived by a reduction from query equivalence.

Historical Background

The SQL standard defines five aggregate functions, namely, `count`, `sum`, `min`, `max` and `avg`(average). Over time, it has become apparent that users would like to aggregate data in additional ways. Therefore, major database systems have added new built-in aggregate functions to meet this need. In addition, many database systems now allow the user to extend the set of available aggregate functions by defining his own aggregate functions.

Aggregate queries are typically used to summarize detailed information. For example, consider a database with the relations `Dept(deptId, deptName)` and `Emp(empId, deptId, salary)`. The following SQL query returns the number of employees, and the total department expenditure on salaries, for each department which has an average salary above $10,000.

```
(Q1) SELECT deptID, count(empID),
            sum(salary)
     FROM Dept, Emp
     WHERE Dept.deptID = Emp.deptID
     GROUP BY Dept.deptID
     HAVING avg(salary) > 10000
```

Typically, aggregate queries have three special components. First, the GROUP BY clause is used to state how intermediate tuples should be grouped before applying aggregation. In this example, tuples are grouped by their value of deptID, i.e., all tuples with the same value for this attribute form a single group. Second, a HAVING clause can be used to determine which groups are of interest, e.g., those with average salary above $10,000. Finally, the outputted aggregate functions are specified in the SELECT clause, e.g., the number of employees and the sum of salaries.

The inclusion of aggregation in SQL has motivated the study of aggregation in relational algebra, as an abstract modeling of SQL. One of the earliest studies of aggregation was by Klug [11], who extended relational algebra and relational calculus to allow aggregate functions and showed the equivalence of these two languages. Aggregation has also been added to Datalog. This has proved challenging since it is not obvious what semantics should be adopted in the presence of recursion [15].

Foundations

Expressiveness

The study of query expressiveness deals with determining what can be expressed in a given query language. The expressiveness of query languages with aggregation has been studied both for the language of relational algebra, as well as for datalog, which may have recursion.

Various papers have studied the expressive power of nonrecursive languages, extended with aggregation, e.g., [7,9,13]. The focus here will be on [12], which has the cleanest, general proofs for the expressive power of languages modeling SQL.

In [12], the expressiveness of variants of relational algebra, extended with aggregation, was studied. First, [12] observes that the addition of aggregation to relational algebra strictly increases its expressiveness. This is witnessed by the query Q2:

```
(Q2) SELECT 1
     FROM R1
     WHERE (SELECT COUNT(*) FROM R) >
           (SELECT COUNT(*) FROM S)
```

Observe that Q2 returns 1 if R contains more tuples than S, and otherwise an empty answer. It is known that first-order logic cannot compare cardinalities, and hence neither can relational algebra. Therefore, SQL with aggregation is strictly more expressive than SQL without aggregation.

The language ALG_{aggr} is presented in [12]. Basically, ALG_{aggr} is relational algebra, extended by arbitrary aggregation and arithmetic functions. In ALG_{aggr}, non-numerical selection predicates are restricted to using only the equality relation (and not order comparisons). A *purely relational query* is one which is applied only to non-numerical data. It is shown that all purely relational queries in ALG_{aggr} are *local*. Intuitively, the answers to local queries are determined by looking at small portions of the input.

The formal definition of local queries follows. Let D be a database. The *Gaifman graph* $G(D)$ of D is the undirected graph on the values appearing in D, with $(a,b) \in G(D)$ if a and b belong to the same tuple of some relation in D. Let $\vec{a} = (a_1,...,a_k)$ be a tuple of values, each of which appears in D. Let r be an integer, and let $S_r^D(\vec{a})$ be the set of values b such that $dist(a_i, b) \leq r$ in $G(D)$, for some i. The *r-neighborhood* $N_r^D(\vec{a})$ of $\vec{a}$ is a new database in which the relations of D are restricted to contain only the values in $S_r^D(\vec{a})$. Then, $\vec{a}$ and $\vec{b}$ are *(D,r)-equivalent* if there is an isomorphism $h : N_r^D(\vec{a}) \to N_r^D(\vec{b})$ such that $h(\vec{a}) = \vec{b}$. Finally, a q is *local* if there exists a number r such that for all D, if $(\vec{a})$ and $(\vec{b})$ are (D,r)-equivalent, then $\vec{a} \in q(D)$ if and only if $\vec{b} \in q(D)$.

There are natural queries that are not local. For example, transitive closure (also called reachability) is not local. Since all queries in ALG_{aggr} are local, this implies that transitive closure cannot be expressed in ALG_{aggr}.

In addition to ALG_{aggr}, [12] introduces the languages $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{N}}$ and $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{Q}}$. $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{N}}$ and $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{Q}}$ are the extensions of ALG_{aggr} which allow order comparisons in the selection predicates, and allow natural numbers and rational numbers, respectively, in the database. It is not known whether transitive closure can be expressed in $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{N}}$. More precisely, [12] shows that if transitive closure is not expressible in $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{N}}$, then the complexity class Uniform TC^0 is properly contained in the complexity class NLOGSPACE. Since the latter problem (i.e., determining strict containment of TC^0 in NLOGSPACE) is believed

to be very difficult to prove, so is the former. Moreover, this result holds even if the arithmetic functions are restricted to $\{+,\cdot,<,0,1\}$ and the aggregate functions are restricted to $\{\texttt{sum}\}$. On the other hand, $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{Q}}$ extended by arbitrary aggregation and arithmetic functions, can express all computable queries.

The languages ALG_{aggr}, $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{N}}$ and $\text{ALG}_{\text{aggr}}^{\leq,\mathbb{Q}}$ are based on relational algebra, and therefore, do not allow recursion. The Datalog language allows queries to be defined as programs, containing recursion. The meaning of an aggregate function within a recursive program, is not always well-defined. One solution is to restrict the program to have only *stratified aggregation*. Stratification means that if a derived predicate p is defined by applying an aggregate function on a derived predicate q, then the definition of q does not depend, syntactically, upon the definition of p. For example, consider the following Datalog program, P_1.

$$\begin{aligned} p(X, \texttt{sum}(Y)) &\leftarrow q(X, Y) \\ q(X, Y) &\leftarrow a(X, Y) \\ q(X, Z) &\leftarrow q(X, Y), q(Y, Z) \end{aligned}$$

The program P_1 is stratified. Replacing the final rule in P_1 with

$$q(X, Z) \leftarrow q(X, Y), p(Y, Z)$$

would yield a program with nonstratified aggregation.

The expressiveness of stratified aggregation was studied in [14]. Only the aggregate functions `sum`, `avg`, `min`, `max` and `count` were allowed. It is shown that that stratified aggregation cannot express *summarized explosion* (i.e., the number of instances of a part needed to construct a bigger part). On the other hand, if the language is extended to allow the function +, as well as the constants 0 and 1, then all computable queries on the integer domain can be expressed. This is correct even if the only aggregate function allowed is `max`. Additional results of this type, i.e., expressibility of other fragments of stratified Datalog, also appear in [14].

Query Containment

The equivalence and containment problems for aggregate queries have been studied for nonrecursive Datalog programs. A survey of the containment and equivalence problems for aggregate queries, containing references to most works on this topic, appears in [2].

Deriving general characterizations of containment (or equivalence) for aggregate queries is difficult, since each aggregate function tends to have its own idiosyncrasies. For example, `count` is sensitive to the number of occurrences of each value, but not to the values themselves, whereas `max` ignores repeated values, but is sensitive to the exact values appearing. As another example, `sum` ignores the value 0, whereas `prod` ignores 1. In addition, `prod` always returns 0 if it is applied to a bag containing 0.

Due to aggregate function quirks, it is often the case that equivalent queries are no longer so, if the aggregate function appearing in their head changes. To demonstrate, consider the two pairs of queries q_1, q_1' and q_2, q_2'.

$$\begin{aligned} q_1(X, \texttt{count}) &\leftarrow a(X, Y) \\ q_1'(X, \texttt{count}) &\leftarrow a(X, Y), a(X, Z) \\ q_2(X, \texttt{max}(Y)) &\leftarrow a(X, Y) \\ q_2'(X, \texttt{max}(Y)) &\leftarrow a(X, Y), a(X, Z) \end{aligned}$$

The queries q_1 and q_2 (and similarly q_1' and q_2') have the same conditions in their body, and differ only on the output aggregate function. One may show that q_1 is not equivalent to q_1' (nor is there containment in either direction), as witnessed by the database

$$D_1 = \{a(\texttt{c}, 0), a(\texttt{c}, 1), a(\texttt{d}, 0)\}$$

over which $q_1(D_1) = \{(\texttt{c},2),(\texttt{d},1)\}$ and $q_1'(D_1) = \{(\texttt{c}, 4),(\texttt{d},1)\}$. On the other hand, $q_2 \equiv q_2'$ does hold.

The different oddities of aggregate functions make finding a general solution for the equivalence and containment problems very difficult. Thus, characterizations for equivalence of aggregate queries often are defined separately for each aggregate function. Most known characterizations for equivalence are based on checking for the existence of special types of mappings between the queries. For example, conjunctive queries (i.e., Datalog programs consisting of a single rule, and no negation) with the aggregate function `count`, are equivalent if and only if they are isomorphic [1,4].

For other types of aggregate functions, as well as for `count` queries with comparisons or disjunctions, isomorphism is not a necessary condition for equivalence. To demonstrate, each pair of queries q_i, q_i' below is equivalent, yet not isomorphic:

$$q_3(\texttt{count}) \leftarrow b(X), b(Y), b(Z), X < Y, X < Z$$
$$q_3'(\texttt{count}) \leftarrow b(X), b(Y), b(Z), X < Z, Y < Z$$

$$q_4(\texttt{sum}(Y)) \leftarrow b(Y), b(Z), Y > 0, Z > 0$$
$$q_4'(\texttt{sum}(Y)) \leftarrow b(Y), b(Z), Y \geq 0, Z > 0$$

$$q_5(\texttt{avg}(Y)) \leftarrow b(Y)$$
$$q_5'(\texttt{avg}(Y)) \leftarrow b(Y), b(Z)$$

$$q_6(\texttt{max}(Y)) \leftarrow b(Y), b(Z_1), b(Z_2), Z_1 < Z_2$$
$$q_6'(\texttt{max}(Y)) \leftarrow b(Y), b(Z), Z < Y$$

Characterizations for equivalence are known for queries of the above types. Specifically, characterizations have been presented for equivalence of conjunctive queries with the aggregate functions `count`, `sum`, `max` and `count-distinct` [4] and these were extended in [5] to queries with disjunctive bodies. Equivalence of conjunctive queries with `avg` and with `percent` were characterized in [8].

It is sometimes possible to define classes of aggregate functions and then present general characterizations for equivalence of queries with any aggregate function within the class of functions. Such characterizations are often quite intricate since they must deal with many different aggregate functions. A characterization of this type was given in [6] to decide equivalence of aggregate queries with *decomposable* aggregate functions, even if the queries contain negation. Intuitively, an aggregate function is decomposable if partially computed values can easily be combined together to return the result of aggregating an entire multiset of values, e.g., as is the case for `count`, `sum` and `max`.

Interestingly, when dealing with aggregate queries it seems that the containment problem is more elusive than the equivalence problem. In fact, for aggregate queries, containment is decided by reducing to the equivalence problem. A reduction of containment to equivalence is presented for queries with *expandable* aggregate functions in [3]. Intuitively, for expandable aggregate functions, changing the number of occurrences of values in bags B and B' does not affect the correctness of the formula $\alpha(B) = \alpha(B')$, as long as the proportion of each value in each bag remains the same, e.g., as is the case for `count`, `sum`, `max`, `count-distinct` and `avg`.

The study of aggregate queries using the `count` function is closely related to the study of nonaggregate queries evaluated under *bag-set semantics*. Most past research on query containment and equivalence for nonaggregate queries assumed that queries are evaluated under *set semantics*. In set semantics, the output of a query does not contain duplicated tuples. (This corresponds to SQL queries with the `DISTINCT` operator.) Under *bag-set semantics* the result of a query is a multiset of values, i.e., the same value may appear many times. A related semantics is *bag semantics* in which both the database and the query results may contain duplication.

To demonstrate the different semantics, recall the database D_1 defined above. Consider evaluating, over D_1, the following variation of q_1:

$$q_1''(X) \leftarrow a(X, Y)$$

Under set-semantics $q''_1(D_1) = \{(\texttt{c}), (\texttt{d})\}$, and under bag-set semantics $q''_1(D_1) = \{\!\{(\texttt{c}), (\texttt{c}), (\texttt{d})\}\!\}$. Note the correspondence between bag-set semantics and using the `count` function, as in q_1, where `count` returns exactly the number of duplicates of each value. Due to this correspondence, solutions for the query containment problem for queries with the `count` function immediately give rise to solutions for the query containment problem for nonaggregate queries evaluated under bag-set semantics, and vice-versa.

The first paper to directly study containment and equivalence for nonaggregate queries under bag-set semantics was [1], which characterized equivalence for conjunctive queries. This was extended in [4] to queries with comparisons, in [5] to queries with disjunctions and in [6] to queries with negation.

Key Applications

Query Optimization

The ability to decide query containment and equivalence is believed to be a key component in query optimization. When optimizing a query, the database can use equivalence characterizations to remove redundant portions of the query, or to find an equivalent, yet cheaper, alternative query.

Query Rewriting

Given a user query q, and previously computed queries $v_1, \ldots, v_n$, the query rewriting problem is to find a query r that (i) is equivalent to q, and (ii) uses the queries

$v_1,...,v_n$ instead of accessing the base relations. (Other variants of the query rewriting problem have also been studied.) Due to Condition (i), equivalence characterizations are needed to solve the query rewriting problem. Query rewriting is useful as an optimization technique, since it can be cheaper to use past results, instead of evaluating a query from scratch. Integrating information sources is another problem that can be reduced to the query rewriting problem.

Future Directions

Previous work on query containment does not consider queries with `HAVING` clauses. Another open problem is containment for queries evaluated under bag-set semantics. In this problem, one wishes to determine if the bag returned by q is always sub-bag of that returned by q'. (Note that this is different from the corresponding problem of determining containment of queries with `count`, which has been solved.) It has shown [10] that bag-set containment is undecidable for conjunctive queries containing inequalities. However, for conjunctive queries without any order comparisons, determining bag-set containment is still an open problem.

Cross-references

- ▶ Answering Queries using Views
- ▶ Bag Semantics
- ▶ Data Aggregation in Sensor Networks
- ▶ Expressive Power of Query Languages
- ▶ Locality
- ▶ Query Containment
- ▶ Query Optimization (in Relational Databases)
- ▶ Query Rewriting using Views

Recommended Reading

1. Chaudhuri S. and Vardi M.Y. Optimization of *real* conjunctive queries. In Proc. 12th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1993, pp. 59–70.
2. Cohen S. Containment of aggregate queries. ACM SIGMOD Rec., 34(1):77–85, 2005.
3. Cohen S., Nutt W., and Sagiv Y. Containment of aggregate queries. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 111–125.
4. Cohen S., Nutt W., and Sagiv Y. Deciding equivalences among conjunctive aggregate queries. J. ACM, 54(2), 2007.
5. Cohen S., Nutt W., and Serebrenik A. Rewriting aggregate queries using views. In Proc. 18th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1999, pp. 155–166.
6. Cohen S., Sagiv Y., and Nutt W. Equivalences among aggregate queries with negation. ACM Trans. Comput. Log., 6(2):328–360, 2005.
7. Consens M.P. and Mendelzon A.O. Low complexity aggregation in graphlog and datalog. Theor. Comput. Sci., 116(1 and 2): 95–116, 1993.
8. Grumbach S., Rafanelli M., and Tininini L. On the equivalence and rewriting of aggregate queries. Acta Inf., 40(8):529–584, 2004.
9. Hella L., Libkin L., Nurmonen J., and Wong L. Logics with aggregate operators. J. ACM, 48(4):880–907, 2001.
10. Jayram T.S., Kolaitis P.G., and Vee E. The containment problem for real conjunctive queries with inequalities. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2006, pp. 80–89.
11. Klug A.C. Equivalence of relational algebra and relational calculus query languages having aggregate functions. J. ACM, 29(3):699–717, 1982.
12. Libkin L. Expressive power of SQL. Theor. Comput. Sci., 3(296):379–404, 2003.
13. Libkin L. and Wong L. Query languages for bags and aggregate functions. J. Comput. Syst. Sci., 55(2):241–272, 1997.
14. Mumick I.S. and Shmueli O. How expressive is statified aggregation? Ann. Math. Artif. Intell., 15(3–4):407–434, 1995.
15. Ross K.A. and Sagiv Y. Monotonic aggregation in deductive database. J. Comput. Syst. Sci., 54(1):79–97, 1997.

Aggregation-Based Structured Text Retrieval

Theodora Tsikrika
Center for Mathematics and Computer Science, Amsterdam, The Netherlands

Definition

Text retrieval is concerned with the retrieval of documents in response to user queries. This is achieved by (i) representing documents and queries with indexing features that provide a characterisation of their information content, and (ii) defining a function that uses these representations to perform retrieval. *Structured text retrieval* introduces a finer-grained retrieval paradigm that supports the representation and subsequent retrieval of the individual document components defined by the document's *logical structure*. *Aggregation-based structured text retrieval* defines (i) the representation of each document component as the aggregation of the representation of its own information content and the representations of information content of its structurally related components, and

(ii) retrieval of document components based on these (aggregated) representations.

The aim of aggregation-based approaches is to improve retrieval effectiveness by capturing and exploiting the interrelations among the components of structured text documents. The representation of each component's own information content is generated at indexing time. The recursive aggregation of these representations, which takes place at the level of their indexing features, leads to the generation, either at indexing or at query time, of the representations of those components that are structurally related with other components.

Aggregation can be defined in numerous ways; it is typically defined so that it enables retrieval to focus on those document components more specific to the query or to each document's best entry points, i.e., document components that contain relevant information and from which users can browse to further relevant components.

Historical Background

A well-established Information Retrieval (IR) technique for improving the effectiveness of *text retrieval* (i.e., retrieval at the document level) has been the generation and subsequent combination of multiple representations for each document [3]. To apply this useful technique to the *text retrieval* of structured text documents, the typical approach has been to exploit their *logical structure* and consider that the individual representations of their components can act as the different representations to be combined [11]. This definition of the representation of a structured text document as the combination of the representations of its components was also based on the intuitive idea that the information content of each document consists of the information content of its sub-parts [2,6].

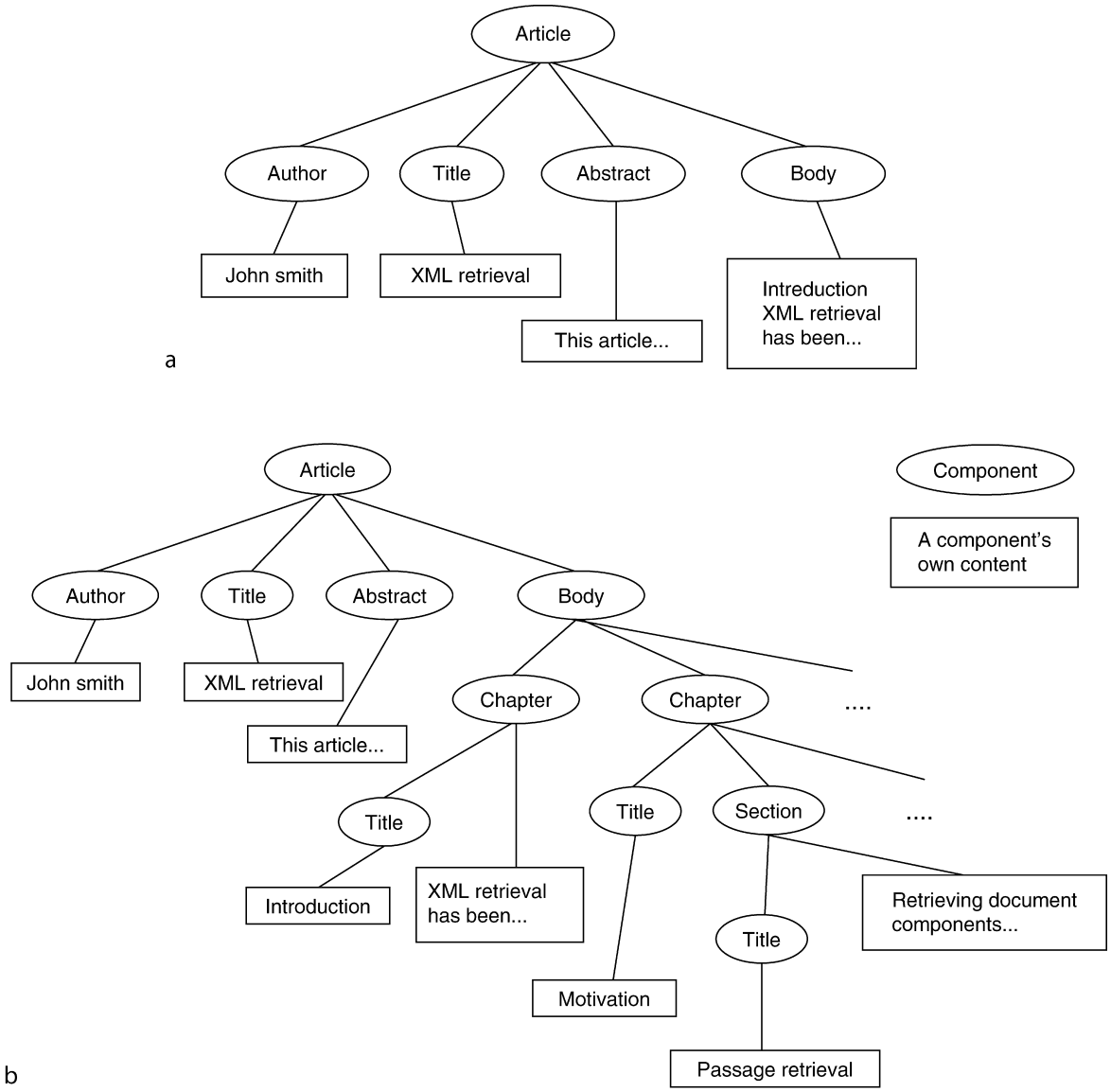
As the above description suggests, these combination-based approaches, despite restricting retrieval only at the document level, assign representations not only to documents, but also to individual document components. To generate these representations, structured text documents can simply be viewed as series of non-overlapping components (Figure 1a), such as title, author, abstract, body, etc. [13]. The proliferation of *SGML* and *XML* documents, however, has led to the consideration of hierarchical components (Figure 1b), and their interrelated representations [1]. For these (disjoint or nested) document components, the combination of their representations can take place (i) directly at the level of their indexing features, which typically correspond to terms and their statistics (e.g., [13]), or (ii) at the level of retrieval scores computed independently for each component (e.g., [15]). Overall, these combination-based approaches have proven effective for the *text retrieval* of structured text documents [11,13,15].

Following the recent shift towards the *structured text retrieval* paradigm [2], which supports the retrieval of document components (including whole documents), it was only natural to try to adapt these combination-based approaches to this new requirement for retrieval at the sub-document level. Here, the focus is on each document component: its representation corresponds to the combination of its own representation with the representations of its structurally related components, and its retrieval is based on this combined representation. Similarly to the case of combination-based approaches for *text retrieval*, two strands of research can be identified: (i) approaches that operate at the level of the components' indexing features (e.g., [12]), referred to as *aggregation-based structured text retrieval* (described in this entry), and (ii) approaches that operate at the level of retrieval scores computed independently for each component (e.g., [14]), referred to as *propagation*-based structured text retrieval.

Figure 2b illustrates the premise of aggregation- and propagated-based approaches for the simple structured text document depicted in Figure 2a. Since these approaches share some of their underlying motivations and assumptions, there has been a cross-fertilisation of ideas between the two. This also implies that this entry is closely related to the entry on *propagation*-based structured text retrieval.

Foundations

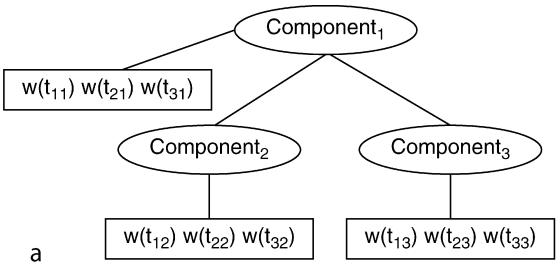
Structured text retrieval supports, in principle, the representation and subsequent retrieval of document components of any granularity; in practice, however, it is desirable to take into account only document components that users would find informative in response to their queries [1,2,4,6]. Such document components are referred to as *indexing units* and are usually chosen (manually or automatically) with respect to the requirements of each application. Once the *indexing*



Aggregation-Based Structured Text Retrieval. Figure 1. Two views on the logical structure of a structured text document.

units have been determined, each can be assigned a representation of its information content, and, hence, become individually retrievable.

Aggregation-based structured text retrieval approaches distinguish two types of *indexing units*: *atomic* and *composite*. Atomic components correspond to *indexing units* that cannot be further decomposed, i.e., the leaf components in Figure 1b. The representation of an atomic component is generated by considering only its own information content. Composite components, on the other hand, i.e., the non-leaf nodes in Figure 1b, correspond to *indexing units* which are related to other components, e.g., consist of sub-components. In addition to its own information content, a composite component is also dependent on the information content of its structurally related components. Therefore, its representation can be derived via the *aggregation* of the representation of its



a

Aggregation: retrieval_score' (component$_1$) = f(w(t$_1$), w(t$_2$), w(t$_3$))

Where:
w(t$_1$) = w(t$_{11}$) ⊕ w(t$_{12}$) ⊕ w(t$_{13}$)
w(t$_2$) = w(t$_{21}$) ⊕ w(t$_{22}$) ⊕ w(t$_{23}$)
w(t$_3$) = w(t$_{31}$) ⊕ w(t$_{32}$) ⊕ w(t$_{33}$)

Propagation: retrieval_score' (component$_1$) = retrieval_score (component$_1$) ⊕ retrieval_score (component$_2$) ⊕ retrieval_score (component$_3$)

Where:
retrieval_score (component$_1$) = f(w(t$_{11}$), w(t$_{21}$), w(t$_{31}$))
retrieval_score (component$_2$) = f(w(t$_{12}$), w(t$_{22}$), w(t$_{32}$))
retrieval_score (component$_3$) = f(w(t$_{13}$), w(t$_{23}$), w(t$_{33}$))

b

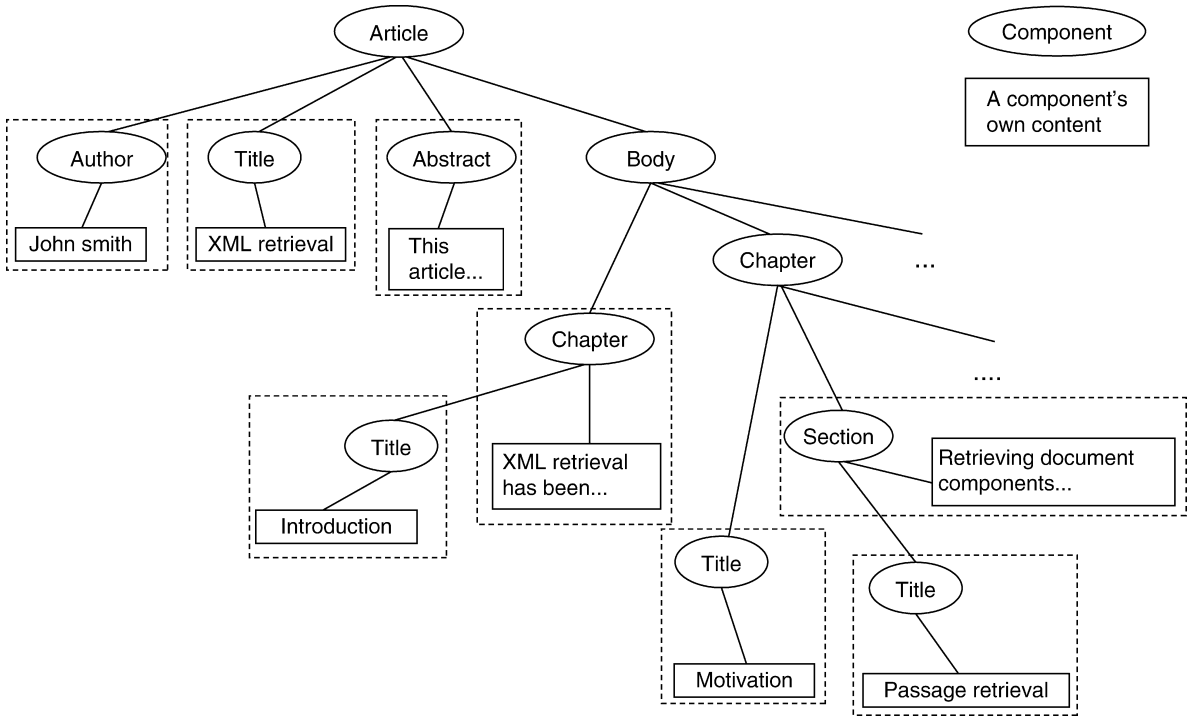
Aggregation-Based Structured Text Retrieval. Figure 2. Simple example illustrating the differences between aggregation- and propagation-based approaches.

own information content with the representations of the information content of its structurally related components; this aggregation takes place at the level of their indexing features. Given the representations of atomic components and of composite components' own information content, aggregation-based approaches recursively generate the aggregated representations of composite components and, based on them, perform retrieval of document components of varying granularity.

In summary, each aggregation-based approach needs to define the following: (i) the representation of each component's own information content, (ii) the aggregated representations of composite components, and (iii) the retrieval function that uses these representations. Although these three steps are clearly interdependent, the major issues addressed in each step need to be outlined first, before proceeding with the description of the key aggregation-based approaches in the field of *structured text retrieval*.

1. *Representing each component's own information content*: In the field of *text retrieval*, the issue of representing documents with indexing features that provide a characterisation of their information content has been extensively studied in the context of several *IR retrieval models* (e.g., Boolean, vector space, probabilistic, language models, etc.). For text documents, these indexing features typically correspond to term statistics. Retrieval functions produce a ranking in response to a user's query, by taking into account the statistics of query terms together with each document's length. The term statistics most commonly used correspond to the *term frequency* $tf(t, d)$ of term t in document d and to the *document frequency* $df(t, C)$ of term t in the document collection C, leading to standard $tf \times idf$ weighting schemes.

Structured text retrieval approaches need to generate representations for all components corresponding to *indexing units*. Since these components are nested, it is not straightforward to adapt these *term statistics* (particularly *document frequency*) at the component level [10]. Aggregation-based approaches, on the other hand, directly generate representations only for components that have their own information content,



Aggregation-Based Structured Text Retrieval. Figure 3. Representing the components that contain their own information.

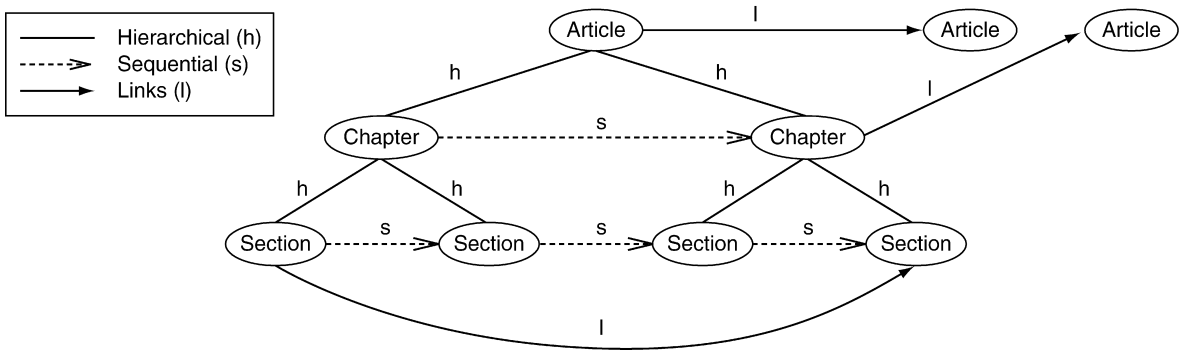
while the representations of the remaining components are obtained via the aggregation process. Therefore, the first step is to generate the representations of atomic components and of the composite components' own information content, i.e., the content not contained in any of their structurally related components. This simplifies the process, since only *disjoint* units need to be represented [6], as illustrated in Figure 3 where the dashed boxes enclose the components to be represented (cf. [5]).

Text retrieval approaches usually consider that the information content of a document corresponds only to its *textual content*, and possibly its metadata (also referred to as *attributes*). In addition to that, *structured text retrieval* approaches also aim at representing the information encoded in the logical structure of documents. Representing this *structural information*, i.e., the interrelations among the documents and their components, enables retrieval in response to both *content-only queries* and *content-and-structure queries*.

Aggregation-based approaches that only represent the textual content typically adapt standard representation formalisms widely employed in *text retrieval* approaches to their requirements for representation at the component level (e.g., [9,11]). Those that consider richer representations of information content apply more expressive formalisms (e.g., various logics [2,4]).

2. *Aggregating the representations*: The concept underlying aggregation-based approaches is that of *augmentation* [4]: the information content of a document component can be *augmented* with that of its structurally related components. Given the already generated representations (i.e., the representations of atomic components and of composite components' own information content), the augmentation of composite components is performed by the aggregation process.

The first step in the aggregation process is the identification of the structurally related components of each composite component. Three basic types of structural relationships (Figure 4) can be distinguished: hierarchical (h), sequential (s), and links (l). Hierarchical connections express the *composition* relationship among components, and induce the tree representing the *logical structure* of a structured



Aggregation-Based Structured Text Retrieval. Figure 4. Different types of structural relationships between the components of a structured text document.

document. Sequential connections capture the order imposed by the document's author(s), whereas links to components of the same or different documents reference (internal or external) sources that offer similar information. In principle, all these types of structural relationships between components can be taken into account by the aggregation process (and some aggregation-based approaches are generic enough to accommodate them, e.g., [7]). In practice, however, the hierarchical structural relations are the only ones usually considered. This leads to the aggregated representations of composite components being recursively generated in an ascending manner.

The next step is to define the *aggregation operator* (or *aggregation function*). Since the aggregation of the textual content of related components is defined at the level of the indexing features of their representations, the aggregation function is highly dependent on the model (formalism) chosen to represent each component's own content. This aggregation results in an (aggregated) representation modeled in the same formalism, and can be seen as being performed at two stages (although these are usually combined into one step): the aggregation of *index expressions* [2] (e.g., terms, conjunctions of terms, etc.), and of the *uncertainty* assigned to them (derived mainly by their statistics).

An aggregation function could also take into account: (i) *augmentation factors* [6], which capture the fact that the textual content of the structurally related components of a composite component is not included in that components own content and has to be "propagated" in order to become part of it, (ii) *accessibility factors* [4], which specify how the representation of a component is influenced by its connected components (a measure of the contribution of, say, a section to its embedding chapter [2]), and (iii) the overall importance of a component in a document's structure [7] (e.g., it can be assumed that a title contains more informative content than a small subsection [13]). Finally, the issue of the possible aggregation of the attributes assigned to related components needs to be addressed [2].

The above aggregation process can take place either at indexing time (*global aggregation*) or at query time (*local aggregation*). Global aggregation is performed for all composite *indexing units* and considers all indexing features involved. Since this strategy does not scale well and can quickly become highly inefficient, local aggregation strategies are primarily used. These restrict the aggregation only to indexing features present in the query (i.e., query terms), and, starting from components retrieved in terms of their own information content, perform the aggregation only for these components' ancestors.

3. *Retrieval*: The retrieval function operates both on the representations of atomic components and on the aggregated representations of composite components. Its definition is highly dependent on the formalism employed in modeling these representations. In conjunction with the definition of the aggregation function, the retrieval function operationalizes the notion of *relevance* for a *structured text retrieval* system. It can, therefore, determine whether retrieval focuses on those document components more specific to the query [2], or whether the aim is to support the users'

browsing activities by identifying each documents best entry points [7] (i.e., document components that contain relevant information which users can browse to further relevant components).

Aggregation-based Approaches

One of the most influential aggregation-based approaches has been developed by Chiaramella et al. [2] in the context of the FERMI project (http://www.dcs.gla.ac.uk/fermi/). Aiming at supporting the integration of IR, hypermedia, and database systems, the FERMI model introduced some of the founding principles of *structured text retrieval* (including the notion of retrieval focussed to the most specific components). It follows the logical view on IR, i.e., it models the retrieval process as inference, and it employs predicate logic as its underlying formalism. The model defines a generic representation of content, attributes, and structural information associated with the *indexing units*. This allows for rich querying capabilities, including support for both *content-only queries* and *content-and-structured queries*. The indexing features of structured text documents can be defined in various ways, e.g., as sets of terms or as logical expressions of terms, while the semantics of the aggregation function depend on this definition. Retrieval can then be performed by a function of the *specificity* of each component with respect to the query.

The major limitation of the FERMI model is that it does not incorporate the uncertainty inherent to the representations of content and structure. To address this issue, Lalmas [8] adapted the FERMI model by using propositional logic as its basis, and extended it by modeling the uncertain representation of the textual content of components (estimated by a $tf \times idf$ weighting scheme) using Dempster-Shafer's theory of evidence. The structural information is not explicitly captured by the formalism; therefore, the model does not provide support for *content-and-structured queries*. The aggregation is performed by Dempster's combination rule, while retrieval is based on the belief values of the query terms.

Fuhr, Gövert, and Rölleke [4] also extended the FERMI model using a combination of (a restricted form of) predicate logic with probabilistic inference. Their model captures the uncertainty in the representations of content, structure, and attributes. Aggregation of index expressions is based on a four-valued logic, allowing for the handling of incomplete information and of inconsistencies arising by the aggregation (e.g., when two components containing contradictory information are aggregated). Aggregation of term weights is performed according to the rules of probability theory, typically by adopting term independence assumptions. This approach introduced the notion of *accessibility factor* being taken into account. Document components are retrieved based on the computed probabilities of query terms occurring in their (aggregated) representations.

Following its initial development in [4], Fuhr and his colleagues investigated further this logic-based probabilistic aggregation model in [5,6]. They experimented with modeling aggregation by different Boolean operators; for instance, they noted that, given terms propagating in the document tree in a bottom-up fashion, a probabilistic-OR function would always result in higher weights for components further up the hierarchy. As this would lead (in contrast to the objectives of *specificity*-oriented retrieval) to the more general components being always retrieved, they introduced the notion of *augmentation factors*. These could be used to "downweight" the weights of terms (estimated by a $tf \times idf$ scheme) that are aggregated in an ascending manner. The effectiveness of their approach has been assessed in the context of the *Initiative for the Evaluation of XML retrieval (INEX)* [6].

Myaeng et al. [11] also developed an aggregation-based approach based on probabilistic inference. They employ Bayesian networks as the underlying formalism for explicitly modeling the (hierarchical) structural relations between components. The document components are represented as nodes in the network and their relations as (directed) edges. They also capture the uncertainty associated with both textual content (again estimated by $tf \times idf$ term statistics) and structure. Aggregation is performed by probabilistic inference, and retrieval is based on the computed beliefs. Although this model allows for document component scoring, in its original publication [11] it is evaluated in the context of *text retrieval* at the document level.

Following the recent widespread application of statistical language models in the field of *text retrieval*, Ogilvie and Callan [8] adapted them to the requirements of *structured text retrieval*. To this end, each document component is modeled by a language model; a unigram language model estimates the probability of a term given

some text. For atomic components, the language model is estimated by their own text by employing a maximum likelihood estimate (MLE). For instance, the probability of term t given the language model θ_T of text T in a component can be estimated by: $P(t|\theta_T) = (1 - \omega)P_{MLE}(t|\theta_T) + \omega P_{MLE}(t|\theta_{collection})$, where ω is a parameter controlling the amount of smoothing of the background collection model. For composite components $comp_i$, the aggregation of language models is modeled as a linear interpolation: $P(t|\theta'_{comp_i}) = \lambda^{c'}_{comp_i} P(t|\theta_{comp_i}) + \sum_{j \in children(comp_i)} \lambda^{c}_{j} P(t|\theta_j)$, where $\lambda^{c'}_{comp_i} + \sum_{j \in children(comp_i)} \lambda^{c}_{j} = 1$. These λs model the contribution of each language model (i.e., document component) in the aggregation, while their estimation is a non-trivial issue. Ranking is typically produced by estimating the probability that each component generated the query string (assuming an underlying multinomial model). The major advantage of the language modeling approach is that it provides guidance in performing the aggregation and in estimating the term weights.

A more recent research study has attempted to apply BM25 (one of the most successful *text retrieval* term weighting schemes) to *structured text retrieval*. Robertson et al. [13] initially adapted BM25 to structured text documents with non-hierarchical components (see Figure 1a), while investigating the effectiveness of retrieval at the document level. Next, they [9] adapted BM25 to deal with nested components (see Figure 1b), and evaluated it in the context of the *INitiative for the Evaluation of XML retrieval (INEX)*.

A final note on these aggregation-based approaches is that most aim at focusing retrieval on those document components more specific to the query. However, there are approaches that aim at modeling the criteria determining what constitutes a best entry point. For instance, Kazai et al. [7] model aggregation as a fuzzy formalisation of linguistic quantifiers. This means that an indexing feature (term) is considered in an aggregated representation of a composite component, if it represents *LQ* of its structurally related components, where *LQ* a linguistic quantifier, such as "at least one," "all," "most," etc. By using these aggregated representations, the retrieval function determines that a component is relevant to a query if *LQ* of its structurally related components are relevant, in essence implementing different criteria of what can be regarded as a best entry point.

Key Applications

Aggregation-based approaches can be used in any application requiring retrieval according to the structured text retrieval paradigm. In addition, such approaches are also well suited to the retrieval of multimedia documents. These documents can be viewed as consisting of (disjoint or nested) components each containing one or more media. Aggregation can be performed by considering atomic components to only contain a single medium, leading to retrieval of components of varying granularity. This was recognized early in the field of structured text retrieval and some of the initial aggregation-based approaches, e.g., [2,4], were developed for multimedia environments.

Experimental Results

For most of the presented approaches, particularly for research conducted in the context of the *INitiative for the Evaluation of XML retrieval (INEX)*, there is an accompanying experimental evaluation in the corresponding reference.

Data Sets

A testbed for the evaluation of structured text retrieval approaches has been developed as part of the efforts of the *INitiative for the Evaluation of XML retrieval (INEX)* (http://inex.is.informatik.uni-duisburg.de/).

URL to Code

The aggregation-based approach developed in [8] has been implemented as part of the open source *Lemur* toolkit (for language modeling and IR), available at: http://www.lemurproject.org/.

Cross-references

- ▶ Content-and-Structure Query
- ▶ Content-Only Query
- ▶ Indexing Units
- ▶ Information Retrieval Models
- ▶ INitiative for the Evaluation of XML Retrieval
- ▶ Logical Structure
- ▶ Propagation-based Structured Text Retrieval
- ▶ Relevance
- ▶ Specificity
- ▶ Structured Document Retrieval
- ▶ Text Indexing and Retrieval

Recommended Reading

1. Chiaramella Y. Information retrieval and structured documents. In Lectures on Information Retrieval, Third European Summer-School, Revised Lectures, LNCS, Vol. 1980. M. Agosti, F. Crestani, and G. Pasi (eds.). Springer, 2001, pp. 286–309.
2. Chiaramella Y., Mulhem P., and Fourel F. A model for multimedia information retrieval. Technical Report FERMI, ESPRIT BRA 8134, University of Glasgow, Scotland, 1996.
3. Croft W.B. Combining approaches to information retrieval. In Advances in Information Retrieval: Recent Research from the Center for Intelligent Information Retrieval, Vol. 7. W.B. Croft (ed.). The Information Retrieval Series, Kluwer Academic, Dordrecht, 2000, pp. 1–36.
4. Fuhr N., Gövert N., and Rölleke T. DOLORES: A system for logic-based retrieval of multimedia objects. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 257–265.
5. Fuhr N. and Großjohann K. XIRQL: A query language for information retrieval in XML documents. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 172–180.
6. Gövert N., Abolhassani M., Fuhr N., and Großjohann K. Content-oriented XML retrieval with HyREX. In Proc. 1st Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, pp. 26–32.
7. Kazai G., Lalmas M., and Rölleke T. A model for the representation and focussed retrieval of structured documents based on fuzzy aggregation. In Proc. 8th Int. Symp. on String Processing and Information Retrieval, 2001, pp. 123–135.
8. Lalmas M. Dempster-Shafer's theory of evidence applied to structured documents: Modelling uncertainty. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 110–118.
9. Lu W., Robertson S.E., and MacFarlane A. Field-weighted XML retrieval based on BM25. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, Revised Selected Papers, LNCS, Vol. 3977, Springer, 2006, pp. 161–171.
10. Mass Y. and Mandelbrod M. Retrieving the most relevant XML components. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 53–58.
11. Myaeng S.-H., Jang D.-H., Kim M.-S., and Zhoo Z.-C. A flexible model for retrieval of SGML documents. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 138–145.
12. Ogilvie P. and Callan J. Hierarchical language models for retrieval of XML components. In Advances in XML Information Retrieval and Evaluation. In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, Revised Selected Papers, LNCS, Vol. 3493, Springer, 2005, pp. 224–237.
13. Robertson S.E., Zaragoza H., and Taylor M. Simple BM25 extension to multiple weighted fields. In Proc. Int. Conf. on Information and Knowledge Management, 2004, pp. 42–49.
14. Sauvagnat K., Boughanem M., and Chrisment C. Searching XML documents using relevance propagation. In Proc. 11th Int. Symp. on String Processing and Information Retrieval, 2004, pp. 242–254.
15. Wilkinson R. Effective retrieval of structured documents. In Proc. 17th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1994, pp. 311–317.

AGMS Sketch

► AMS Sketch

Air Indexes for Spatial Databases

Baihua Zheng
Singapore Management University, Singapore, Singapore

Definition

Air indexes refer to indexes employed in wireless broadcast environments to address scalability issue and to facilitate power saving on mobile devices [4]. To retrieve a data object in wireless broadcast systems, a mobile client has to continuously monitor the broadcast channel until the data arrives. This will consume a lot of energy since the client has to remain active during its waiting time. The basic idea of air indexes is that by including index information about the arrival times of data items on the broadcast channel, mobile clients are able to predict the arrivals of their desired data. Thus, they can stay in power saving mode during waiting time and switch to active mode only when the data of their interests arrives.

Historical Background

In spatial databases, clients are assumed to be interested in data objects having spatial features (e.g., hotels, ATM, gas stations). "Find me the nearest restaurant" and "locate all the ATMs that are within 100 miles of my current location" are two examples. A central server is allocated to keep all the data, based on which the queries issued by the clients are answered. There are basically two approaches to disseminating spatial data to clients: (i) *on-demand access*: a mobile client submits a request, which consists of a query and the query's issuing location, to the server. The server returns the result to the mobile client via a dedicated point-to-point channel. (ii) *periodic broadcast*: data are

periodically broadcast on a wireless channel open to the public. After a mobile client receives a query from its user, it tunes into the broadcast channel to receive the data of interest based on the query and its current location.

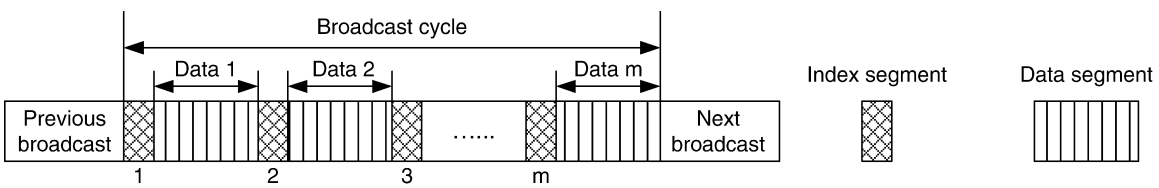
On-demand access is particularly suitable for light-loaded systems when contention for wireless channels and server processing is not severe. However, as the number of users increases, the system performance deteriorates rapidly. Compared with on-demand access, broadcast is a more scalable approach since it allows simultaneous access by an arbitrary number of mobile clients. Meanwhile, clients can access spatial data without reporting to the server their current location and hence the private location information is not disclosed.

In the literature, two performance metrics, namely *access latency* and *tuning time*, are used to measure access efficiency and energy conservation, respectively [4]. The former means the time elapsed between the moment when a query is issued and the moment when it is satisfied, and the latter represents the time a mobile client stays active to receive the requested data. As energy conservation is very critical due to the limited battery capacity on mobile clients, a mobile device typically supports two operation modes: *active mode* and *doze mode*. The device normally operates in active mode; it can switch to doze mode to save energy when the system becomes idle.

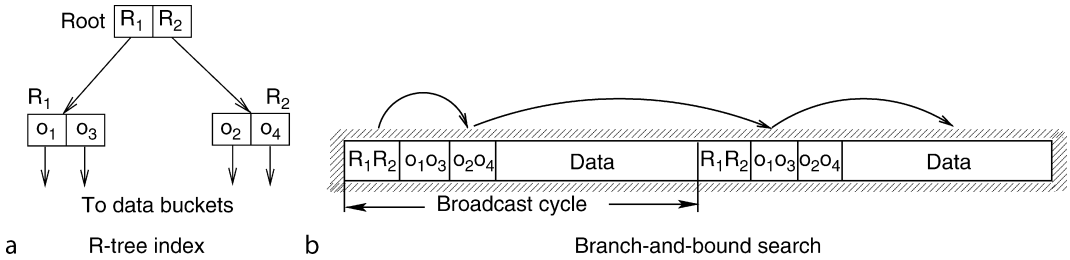
With data broadcast, clients listen to a broadcast channel to retrieve data based on their queries and hence are responsible for query processing. Without any index information, a client has to download all data objects to process spatial search, which will consume a lot of energy since the client needs to remain active during a whole broadcast cycle. A broadcast cycle means the minimal duration within which all the data objects are broadcast at least once. A solution to this problem is *air indexes* [4]. The basic idea is to broadcast an index before data objects (see Fig. 1 for an example). Thus, query processing can be performed over the index instead of actual data objects. As the index is much smaller than the data objects and is selectively accessed to perform a query, the client is expected to download less data (hence incurring less tuning time and energy consumption) to find the answers. The disadvantage of air indexing, however, is that the broadcast cycle is lengthened (to broadcast additional index information). As a result, the access latency would be worsen. It is obvious that the larger the index size, the higher the overhead in access latency.

An important issue in air indexes is how to multiplex data and index on the sequential-access broadcast channel. Figure 1 shows the well-known (1, *m*) scheme [4], where the index is broadcast in front of every 1/m fraction of the dataset. To facilitate the access of index, each data page includes an offset to the beginning of the next index. The general access protocol for processing spatial search involves following three steps: (i) *initial probe*: the client tunes into the broadcast channel and determines when the next index is broadcast; (ii) *index search*: The client tunes into the broadcast channel again when the index is broadcast. It selectively accesses a number of index pages to find out the spatial data object and when to download it; and (iii) *data retrieval*: when the packet containing the qualified object arrives, the client downloads it and retrieves the object.

To disseminate spatial data on wireless channels, well-known spatial indexes (e.g., R-trees) are candidates for air indexes. However, unique characteristics of wireless data broadcast make the adoption of existing spatial indexes inefficient (if not impossible). Specifically, traditional spatial indexes are designed to cluster data objects with spatial locality. They usually assume a resident storage (such as disk and memory) and adopt search strategies that minimize I/O cost. This is achieved by *backtracking* index nodes during search. However, the broadcast order (and thus the access order) of



Air Indexes for Spatial Databases. Figure 1. Air indexes in wireless broadcast environments.



Air Indexes for Spatial Databases. Figure 2. Linear access on wireless broadcast channel.

index nodes is extremely important in wireless broadcast systems because data and index are only available to the client when they are broadcast on air. Clients cannot randomly access a specific data object or index node but have to wait until the next time it is broadcast. As a result, each backtracking operation extends the access latency by one more cycle and hence becomes a constraint in wireless broadcast scenarios.

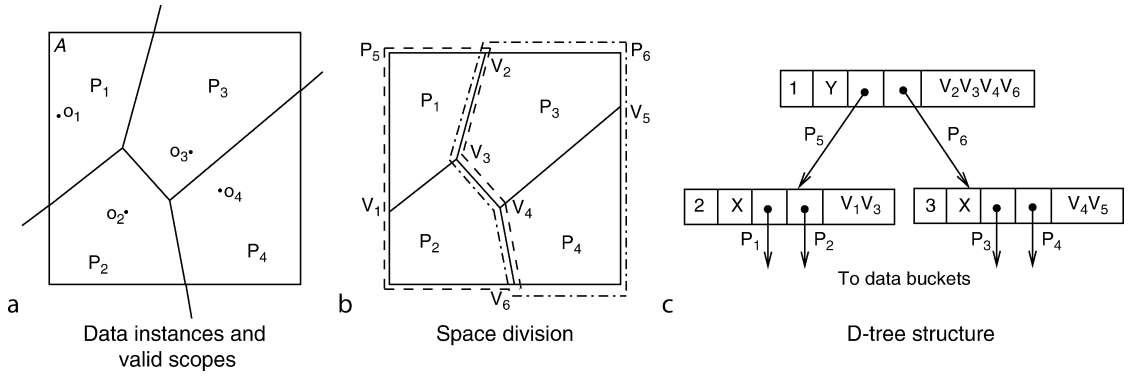
Figure 2 depicts an example of spatial query. Assume that an algorithm based on R-tree first visits root node, then the node R_2, and finally R_1, while the server broadcasts nodes in the order of root, R_1, and R_2. If a client wants to backtrack to node R_1 after it retrieves R_2, it will have to wait until the next cycle because R_1 has already been broadcast. This significantly extends the access latency and it occurs every time a navigation order is different from the broadcast order. As a result, new air indexes which consider both the constraints of the broadcast systems and features of spatial queries are desired.

Foundations

Several air indexes have been recently proposed to support broadcast of spatial data. These studies can be classified into two categories, according to the nature of the queries supported. The first category focuses on retrieving data associated with some specified geographical range, such as "Starbucks Coffee in New York City's Times Square" and "Gas stations along Highway 515." A representative is the index structure designed for *DAYS* project [1]. It proposes a location hierarchy and associates data with locations. The index structure is designed to support query on various types of data with different location granularity. The authors intelligently exploit an important property of the locations, i.e., containment relationship among the objects, to determine the relative location of an object with respect to its parent that contains the object. The containment relationship limits the search range of available data and thus facilitates efficient processing of the supported queries. In brief, a broadcast cycle consists of several sub-cycles, with each containing data belonging to the same type. A major index (one type of index buckets) is placed at the beginning of each sub-cycle. It provides information related to the types of data broadcasted, and enables clients to quickly jump into the right sub-cycle which contains her interested data. Inside a sub-cycle, minor indexes (another type of index buckets) are interleaved with data buckets. Each minor index contains multiple pointers pointing to the data buckets with different locations. Consequently, a search for a data object involves accessing a major index and several minor indexes.

The second category focuses on retrieving data according to specified distance metric, based on client's current location. An example is nearest neighbor (NN) search based on Euclidian distance. According to the index structure, indexes of this category can be further clustered into two groups, i.e., *central tree-based* structure and *distributed* structure. In the following, we review some of the representative indexes of both groups.

D-tree is a paged binary search tree to index a given solution space in support of planar point queries [6]. It assumes a data type has multiple data instances, and each instance has a certain *valid scope* within which this instance is the only correct answer. For example, restaurant is a data type, and each individual restaurant represents an instance. Take NN search as an example, Fig. 3a illustrates four restaurants, namely o_1, o_2, o_3, and o_4, and their corresponding valid scopes p_1, p_2, p_3, and p_4. Given any query location q in, say, p_3, o_3 is the restaurant to which q is nearest. D-tree assumes the valid scopes of different data instances are known and it focuses only on planar point queries which locate the



Air Indexes for Spatial Databases. Figure 3. Index construction using the D-tree.

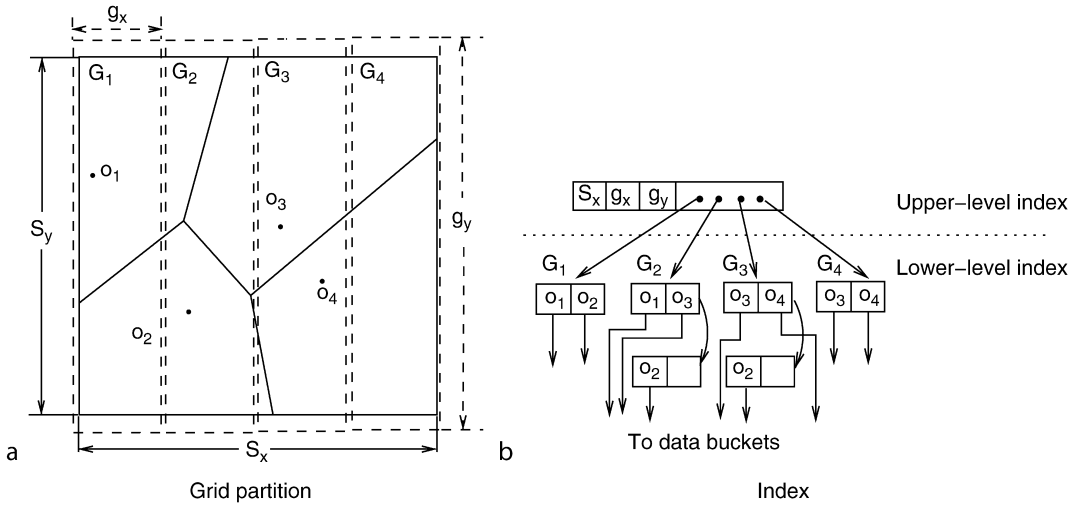
query point into a valid scope and return the client the corresponding data instance.

The D-tree is a binary tree built based on the divisions between data regions (e.g., valid scopes). A space consisting of a set of data regions is recursively partitioned into two complementary subspaces containing about the same number of regions until each subspace has one region only. The partition between two subspaces is represented by one or more polylines. The overall orientation of the partition can be either x-dimensional or y-dimensional, which is obtained, respectively, by sorting the data regions based on their lowest/uppermost y-coordinates, or leftmost/rightmost x-coordinates. Figure 3b shows the partitions for the running example. The polyline $pl(v_2, v_3, v_4, v_6)$ partitions the original space into p_5 and p_6, and polylines $pl(v_1,v_3)$ and $pl(v_4,v_5)$ further partition p_5 into p_1 and p_2, and p_6 into p_3 and p_4, respectively. The first polyline is y-dimensional and the remaining two are x-dimensional. Given a query point q, the search algorithm works as follows. It starts from the root and recursively follows either the left subtree or the right subtree that bounds the query point until a leaf node is reached. The associated data instance is then returned as the final answer.

Grid-partition index is specialized for NN problem [9]. It is motivated by the observation that an object is the NN only to the query points located inside its Voronoi Cell. Let $O = \{o_1, o_2,...,o_n\}$ be a set of points. $V(o_i)$, the *Voronoi cell* (VC) for o_i, is defined as the set of points q in the space such that $dist(q,o_i) < dist(q,o_j)$, $\forall j \neq i$. That is, $V(o_i)$ consists of the set of points for which o_i is the NN. As illustrated in Fig. 3a, p_1, p_2, p_3, and p_4 denote the VCs for four objects, o_1, o_2, o_3, and o_4, respectively. Grid-partition index tries to reduce the search space for a query at the very beginning by partitioning the space into disjoint grid cells. For each grid cell, all the objects that could be NNs of at least one query point inside the grid cell are indexed, i.e., those objects whose VCs overlap with the grid cell are associated with that grid cell.

Figure 4a shows a possible grid partition for the running example, and the index structure is depicted in Fig. 4b. The whole space is divided into four grid cells; i.e., G_1, G_2, G_3, and G_4. Grid cell G_1 is associated with objects o_1 and o_2, since their VCs, p_1 and p_2, overlap with G_1; likewise, grid cell G_2 is associated with objects o_1, o_2, o_3, and so on. If a given query point is in grid cell G_1, the NN can be found among the objects associated with G_1 (i.e., o_1 and o_2), instead of among the whole set of objects. Efficient search algorithms and partition approaches have been proposed to speed up the performance.

Conventional spatial index R-tree has also been adapted to support kNN search in broadcast environments [2]. For R-tree index, the kNN search algorithm would visit index nodes and objects sequentially as backtracking is not feasible on the broadcast. This certainly results in a considerably long tuning time especially when the result objects are located in later part of the broadcast. However, if clients know that there are at least k objects in the later part of the broadcast that are closer to the query point than the currently found ones, they can safely skip the downloading of the intermediate objects currently located. This observation motivates the design of the enhanced kNN search algorithm which caters for the constraints of wireless broadcast. It requires each index node to

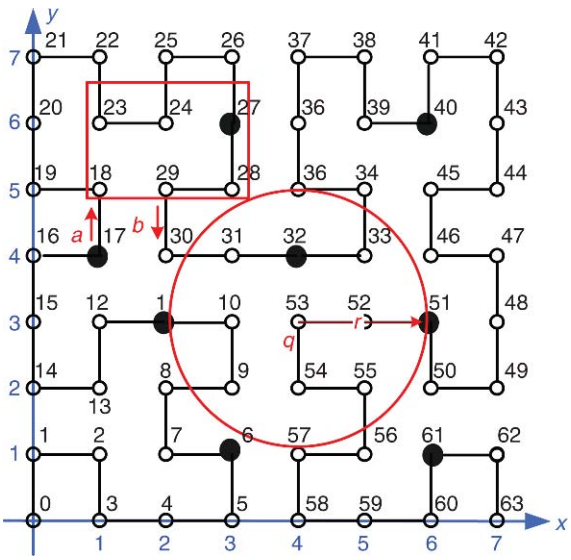


Air Indexes for Spatial Databases. Figure 4. Index construction using the grid-partition.

carry a count of the underlying objects (object count) referenced by the current node. Thus, clients do not blindly download intermediate objects.

Hilbert Curve Index (HCI) is designed to support general spatial queries, including window queries, *k*NN queries, and continuous nearest-neighbor (CNN) queries in wireless broadcast environments. Motivated by the linear streaming property of the wireless data broadcast channel and the optimal spatial locality of the Hilbert Curve (HC), HCI organizes data according to Hilbert Curve order [7,8], and adopts B^+-tree as the index structure. Figure 5 depicts a 8×8 grid, with solid dots representing data objects. The numbers next to the data points, namely *index value*, represent the visiting orders of different points at Hilbert Curve. For instance, data point with (1,1) as the coordinates has the index value of 2, and it will be visited before data point with (2,2) as the coordinates because of the smaller index value.

The *filtering and refining* strategy is adopted to answer all the queries. For window query, the basic idea is to decide a candidate set of points along the Hilbert curve which includes all the points within the query window and later to filter out those outside the window. Suppose the rectangle shown in Fig. 5 is a query window. Among all the points within the search range, the first point is point *a* and the last is *b*, sorted according to their occurring orders on the Hilbert curve, and both of them are lying on the boundary of the search range. Therefore, all the points inside this query window should lie on the Hilbert curve segmented by



Air Indexes for Spatial Databases. Figure 5. Hilbert curve index.

points *a* and *b*. In other words, data points with index values between 18 and 29, but not the others, are the candidates. During the access, the client can derive the coordinates of data points based on the index values and then retrieve those within the query window.

For *k*NN query, the client first retrieves those *k* nearest objects to the query point along the Hilbert curve and then derives a range which for sure bounds at least *k* objects. In the filtering phase, a window query

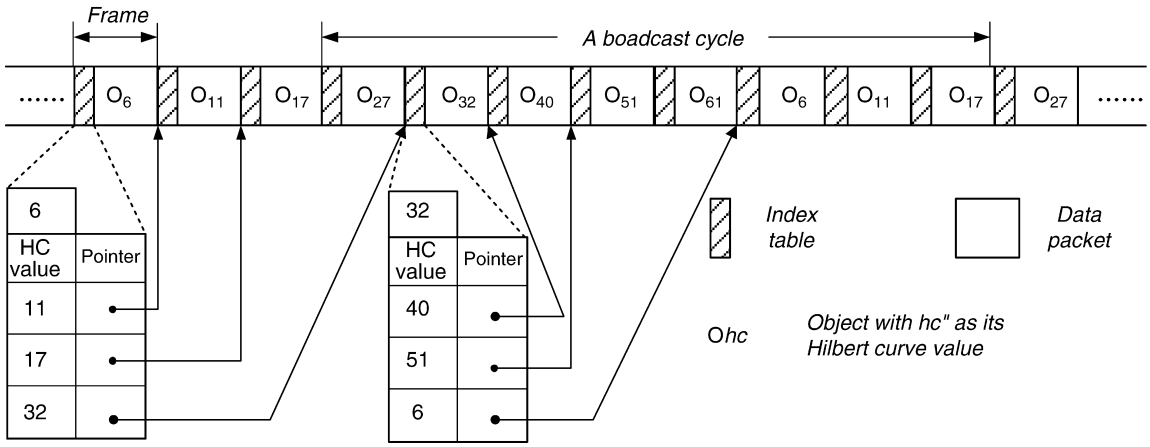
which bounds the search range is issued to filter out those unqualified. Later in the refinement phase, k nearest objects are identified according to their distance to the query point. Suppose an NN query at point q (i.e., index value 53) is issued. First, the client finds its nearest neighbor (i.e., point with index value 51) along the curve and derives a circle centered at q with r as the radius (i.e., the green circle depicted in Fig. 5). Since the circle bounds point 51, it is certain to contain the nearest neighbor to point q. Second, a window query is issued to retrieve all the data points inside the circle, i.e., points with index values 11, 32, and 51. Finally, the point 32 is identified as the nearest neighbor. The search algorithm for CNN adopts a similar approach. It approximates a search range which is guaranteed to bound all the answer objects, issues a window query to retrieve all the objects inside the search range, and finally filters out those unqualified.

All the indexes mentioned above are based on a central tree-based structure, like R-tree and B-tree. However, employing a tree-based index on a linear broadcast channel to support spatial queries results in several deficiencies. First, clients can only start the search when they retrieve the root node in the channel. Replicating the index tree in multiple places in the broadcast channel provides multiple search starting points, shortening the initial root-probing time. However, a prolonged broadcast cycle leads to a long access latency experienced by the clients. Second, wireless broadcast media is not error-free. In case of losing intermediate nodes during the search process, the clients are forced to either restart the search upon an upcoming root node or scan the subsequential broadcast for other possible nodes in order to resume the search, thus extending the tuning time. *Distributed spatial index (DSI)*, a fully distributed spatial index structure, is motivated by these observations [5]. A similar distributed structure was proposed in [3] as well to support access to spatial data on air.

DSI is very different from tree-based indexes, and is not a hierarchical structure. Index information of spatial objects is fully distributed in DSI, instead of simply replicated in the broadcast. With DSI, the clients do not need to wait for a root node to start the search. The search process launches immediately after a client tunes into the broadcast channel and hence the initial probe time for index information is minimized. Furthermore, in the event of data loss, clients resume the search quickly.

Like HCI, DSI also adopts Hilbert curve to determine broadcast order of data objects. Data objects, mapped to point locations in a 2-D space, are broadcast in the ascending order of their HC index values. Suppose there are N objects in total, DSI chunks them into n_F *frames*, with each having n_o objects ($n_F = \lceil N/n_o \rceil$). The space covered by Hilbert Curve shown in Fig. 5 is used as a running example, with solid dots representing the locations of data objects (i.e., $N = 8$). Figure 6 demonstrates a DSI structure with n_o set to 1, i.e., each frame contains only one object.

In addition to objects, each frame also has an index table as its header, which maintains information



Air Indexes for Spatial Databases. Figure 6. Distributed spatial index.

regarding to the HC values of data objects to be broadcast with specific waiting interval from the current frame. This waiting interval can be denoted by delivery time difference or number of data frames apart, with respect to the current frame. Every index table keeps n_i entries, each of which, τ_j, is expressed in the form of $\langle HC'_j, P_j \rangle$, $j \in [0, n_i)$. P_j is a pointer to the r^j-th frame after the current frame, where r (> 1) is an exponential base (i.e., a system-wide parameter), and HC'_j is the HC value of the first object inside the frame pointed by P_j. In addition to τ_j, an index table also keeps the HC values HC_k ($k \in [1, n_o]$) of all the objects obj_k that are contained in the current frame. This extra information, although occupying litter extra bandwidth, can provide a more precise image of all the objects inside current frame. During the retrieval, a client can compare HC_ks of the objects against the one she has interest in, so the retrieval of unnecessary object whose size is much larger than an HC value can be avoided.

Refer to the example shown in Fig. 5, with corresponding DSI depicted in Fig. 6. Suppose $r = 2$, $n_o = 1$, $n_F = 8$, and $n_i = 3$. The index tables corresponding to frames of data objects O_6 and O_{32} are shown in the figure. Take the index table for frame O_6 as an example: τ_0 contains a pointer to the next upcoming (2^0-th) frame whose first object's HC value is 11, τ_1 contains a pointer to the second (2^1-th) frame with HC value for the first object (the only object) 17, and the last entry τ_2 points to the fourth (2^2-th) frame. It also keeps the HC value 6 of the object O_6 in the current frame. Search algorithm for window queries and kNN searches are proposed.

Key Applications

Location-based Service

Wireless broadcast systems, because of the scalability, provide an alternative to disseminate location-based information to a large number of users. Efficient air indexes enable clients to selectively tune into the channel and hence the power consumption is reduced.

Moving Objects Monitoring

Many moving objects monitoring applications are interested in finding out all the objects that currently satisfy certain conditions specified by the users. In many cases, the number of moving objects is much larger than the number of submitted queries. As a result, wireless broadcast provides an ideal way to deliver subscribed queries to the objects, and those objects that might affect the queries can then report their current locations.

Cross-references

- ▶ Nearest Neighbor Query
- ▶ Space-Filling Curves for Query Processing
- ▶ Spatial Indexing Techniques
- ▶ Voronoi Diagrams

Recommended Reading

1. Acharya D. and Kumar V. Location based indexing scheme for days. In Proc. 4th ACM Int. Workshop on Data Eng. for Wireless and Mobile Access, 2005, pp. 17–24.
2. Gedik B., Singh A., and Liu L. Energy efficient exact knn search in wireless broadcast environments. In Proc. 12th ACM Int. Symp. on Geographic Inf. Syst., 2004, pp. 137–146.
3. Im S., Song M., and Hwang C. An error-resilient cell-based distributed index for location-based wireless broadcast services. In Proc. 5th ACM Int. Workshop on Data Eng. for Wireless and Mobile Access, 2006, pp. 59–66.
4. Imielinski T., Viswanathan S., and Badrinath B.R. Data on air – organization and access. IEEE Trans. Knowl. Data Eng., 9(3):1997.
5. Lee W.-C. and Zheng B. Dsi: a fully distributed spatial index for wireless data broadcast. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2005, pp. 349– 358.
6. Xu J., Zheng B., Lee W.-C., and Lee D.L. The d-tree: an index structure for location-dependent data in wireless services. IEEE Trans. Knowl. Data Eng., 16(12):1526–1542, 2002.
7. Zheng B., Lee W.-C., and Lee D.L. Spatial queries in wireless broadcast systems. ACM/Kluwer J. Wireless Networks, 10 (6):723–736, 2004.
8. Zheng B., Lee W.-C., and Lee D.L. On searching continuous k nearest neighbors in wireless data broadcast systems. IEEE Trans. Mobile Comput., 6(7):748–761, 2007.
9. Zheng B., Xu J., Lee W.-C., and Lee L. Grid-partition index: a hybrid method for nearest-neighbor queries in wireless location-based services. VLDB J., 15(1):21–39, 2006.

AJAX

Alex Wun
University of Toronto, Toronto, ON, Canada

Definition

AJAX is an acronym for "Asynchronous JavaScript and XML" and refers to a collection of web development

technologies used together to create highly dynamic web applications.

Key Points

AJAX does not refer to a specific technology, but instead refers to a collection of technologies used in conjunction to develop dynamic and interactive web applications. The two main technologies comprising AJAX are the JavaScript scripting language and the W3C open standard XMLHttpRequest object API. While the use of XML and DOM are important for standardized data representation, using neither XML nor DOM is required for an application to be considered AJAX-enabled since the XMLHttpRequest API actually supports any text format.

Using the XMLHttpRequest API, web applications can fetch data asynchronously while registering a callback function to be invoked once the fetched data is available. More concretely, the XMLHttpRequest object issues a standard HTTP POST or GET request to a web server but returns control to the calling application immediately after issuing the request. The calling application is then free to continue execution while the HTTP request is being handled on the server. When the HTTP response is received, the XMLHttpRequest object calls back into the function that was supplied by the calling application so that the response can be processed. The asynchronous callback model used in AJAX applications is analogous to the Operating System technique of using interrupt handlers to avoid blocking on I/O. As such, development using AJAX necessarily requires an understanding of multi-threaded programming.

There are three main benefits to using AJAX in web applications:

1. *Performance*: Since XMLHttpRequest calls are asynchronous, client-side scripts can continue execution after issuing a request without being blocked by potentially lengthy data transfers. Consequently, web pages can be easily populated with data fetched in small increments in the background.
2. *Interactivity*: By maintaining long-lived data transfer requests, an application can closely approximate real-time event-driven behavior without resorting to periodic polling, which can only be as responsive as the polling frequency.
3. *Data Composition*: Web applications can easily pull data from multiple sources for aggregation and processing on the client-side without any dependence on HTML form elements. Data composition is also facilitated by having data adhere to standard XML and DOM formats.

The functionality provided by AJAX allows web applications to appear and behave much more like traditional desktop applications. The main difference is that data consumed by the application resides primarily out on the Internet – one of the concepts behind applications that are labeled as being representative "Web 2.0" applications.

Cross-references

- ▶ JavaScript
- ▶ MashUp
- ▶ Web 2.0/3.0
- ▶ XML

Recommended Reading

1. The Document Object Model: W3C Working Draft. Available at: http://www.w3.org/DOM/
2. The XMLHttpRequest Object: W3C Working Draft. Available at: http://www.w3.org/TR/XMLHttpRequest/

Allen's Relations

Peter Revesz[1], Paolo Terenziani[2]
[1]University of Nebraska-Lincoln, Lincoln, NE, USA
[2]University of Turin, Turin, Italy

Synonyms

Qualitative relations between time intervals; Qualitative temporal constraints between time intervals

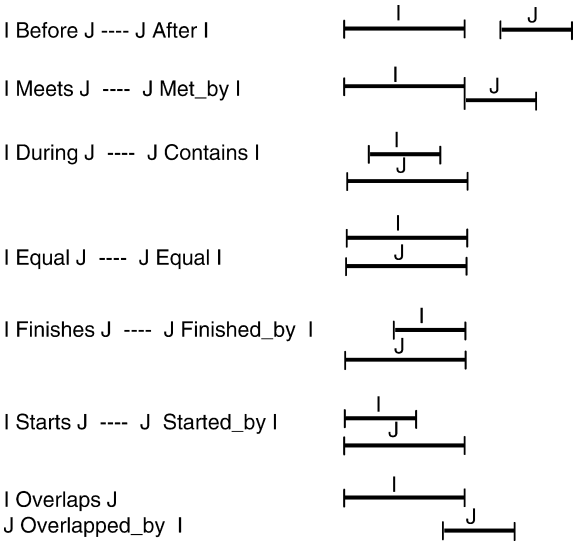
Definition

A (convex) *time interval* I is the set of all time points between a starting point (usually denoted by I^-) and an ending point (I^+). Allen's relations model all possible relative positions between two time intervals [1]. There are 13 different possibilities, depending on the relative positions of the endpoints of the intervals (Table 1).

For example, "There will be a guest speaker during the Database System class" can be represented by Allen's relation I_{Guest} *During* $I_{Database}$ (or by $I^-_{Guest} > I^-_{Database} \wedge I^+_{Guest} < I^+_{Database}$ considering the relative

Allen's Relations. Table 1. Translation of Allen's interval relations between two intervals I and J into conjunctions of point relations between I−, I+, J−, J+.

	$I^- J^-$	$I^- J^+$	$I^+ J^-$	$I^+ J^+$
After		>		
Before			<	
Meets			=	
Met_by		=		
During	>			<
Contains	<			>
Equal	=			=
Finishes	>			=
Finished_by	<			=
Starts	=			<
Started_by	=			>
Overlaps	<		>	<
Overlapped_by	>	<		>



Allen's Relations. Figure 1. Visualization of Allen's interval relations.

position of the endpoints. Moreover, any subset of the 13 relations, excluding the empty subset, is a relation in Allen's Interval Algebra (therefore, there are 2^{13}-1 relations in Allen's Algebra). Such subsets are used in order to denote ambiguous cases, in which the relative position of two intervals is only partially known. For instance, I_1 (Before, Meets, Overlaps) I_2 represents the fact that I_1 is before *or* meets *or* overlaps I_2.

Key Points

In many cases, the exact time interval when facts occur is not known, but (possibly imprecise) information on the relative temporal location of facts is available. Allen's relations allow one to represent such cases of *temporal indeterminacy*. For instance, planning in Artificial Intelligence is the first application of Allen's relations. A graphical representation of the basic 13 Allen's relations is shown in Fig. 1.

Allen's relations are specific cases of *temporal constraints*. Namely, they are qualitative temporal constraints between time intervals. Given a set of such constraints, *qualitative temporal reasoning* can be used in order to make inferences (e.g., to check whether the set of constraints is consistent).

Finally, notice that, in many entries of this Encyclopedia, the term *(time) period* has been used with the same meaning of *(time) interval* in this entry.

Cross-references

- ▶ Qualitative Temporal Reasoning
- ▶ Temporal Constraints
- ▶ Temporal Indeterminacy

Recommended Reading

1. Allen J.F. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11):832–843, 1983.

AMOSQL

Peter M. D. Gray
University of Aberdeen, Aberdeen, UK

Definition

AMOSQL [2] is a functional language having its roots in the functional query languages OSQL [1] and DAPLEX [5] with extensions of mediation primitives, multi-directional foreign functions, late binding, *active rules*, etc. Queries are specified using the select–from–where construct as in SQL. Furthermore, AMOSQL has aggregation operators, nested subqueries, disjunctive queries, quantifiers, and is relationally complete.

AMOSQL is a functional query language operating within the environment of Amos II , which is an open,

light-weight, and extensible database management system (DBMS) with a functional data model. Each Amos II server contains all the traditional database facilities, such as a storage manager, a recovery manager, a transaction manager, and a query language. The system can be used as a single-user database or as a multi-user server to applications and to other Amos II peers. It has mainly been used for experiments with Mediators [4].

Key Points

AMOSQL is often used within a distributed mediator system where several mediator peers communicate over the Internet. Functional views provide transparent access to data sources from clients and other mediator peers. Amos II mediators are composable since a mediator peer can regard other mediator peers as data sources. In AMOSQL [2] the top-level language uses function application within an SQL-like syntax, for example to list children of a particular parent who like sailing:

```
create function sailch(person p) -> string as
select name(c) from person c
where parent(c) = p and hobby(c) = 'sailing';
```

This is turned internally into a typed object comprehension.

```
sailch(p) == [name(c) | c <- person; parent(c)=p; hobby(c) = 'sailing']
```

Some functions may be defined as views on other databases accessed through mediators. In the comprehension form, such functions may easily be substituted because of referential transparency, leading to a longer conjunction with extra clauses [3].

Suppose information about `hobby(c)` was held in connection with `sports-person`, a subclass of person with a `surname` attribute:

```
hobby(c) == [recreation(x) | x <- sports-person; surname(x) = name(c)]
```

These comprehensions merge into the following which can be further simplified:

```
sailch(p) == [name(c) | c <- person; parent(c)=p; x <- sportsperson;
surname(x) = name(c); recreation(x) = 'sailing']
```

Note that the internal form of comprehension does not explicitly distinguish the use of generators but simply use equality as in a filter. The conceptual advantage of this is that generators are not always a fixed role and some optimizations may reverse the role of filter and generator (systems with explicit generators use rewrite rules to do this).

Cross-references

- ▶ Comprehensions
- ▶ Functional Query Language

Recommended Reading

1. Beech D. A foundation of evolution from relational to object databases. In Advances in Database Technology, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp 251–270.
2. Fahl G., Risch T., and Sköld M. 1AMOS – an architecture for active mediators. In Proc. Workshop on Next Generation Information Technologies and Systems, 1993.
3. Josifovski V. and Risch T. Functional query optimization over object-oriented views for data integration. J. Intell. Inf. Syst., 12 (2–3):165–190, 1999.
4. Risch T., Josifovski V., and Katchaounov. T. Functional Data Integration in a Distributed Mediator System. In The Functional Approach to Data Management, chapter 9, P.M.D. Gray, L. Kerschberg, P.J.H. King, and A. Poulovassilis (eds.). Springer, Berlin Heidelberg New York, 2004.
5. Shipman D.W. The functional data model and the data language DAPLEX. ACM Trans. Database Syst., 6(1):140–173, 1981.

AMS Sketch

Alin Dobra
University of Florida, Gainesville FL, USA

Synonyms

AGMS sketch; Sketch; Tug-of-war sketch

Definition

AMS sketches are randomized summaries of the data that can be used to compute aggregates such as the second frequency moment (the self-join size) and sizes of joins. AMS sketches can be viewed as random projections of the data in the frequency domain on ± 1 pseudo-random vectors. The key property of AMS

sketches is that the product of projections on the same random vector of frequencies of the join attribute of two relations is an unbiased estimate of the size of join of the relations. While a single AMS sketch is inaccurate, multiple such sketches can be computed and combined using averages and medians to obtain an estimate of any desired precision.

Historical Background

The AMS sketches were introduced in 1996 by Noga Alon, Yossi Matias, and Mario Szegedy as part of a suit of randomized algorithms for approximate computation of frequency moments. The same authors, together with Phillip Gibbons, extended the second frequency moment application of AMS sketches to the computation of the size of join of two relations, a more relevant database application. The initial work on AMS sketches fostered a large amount of subsequent work on data streaming algorithms including generalizations and extensions of AMS sketches. Alon, Matias, and Szegedy received the Gödel Prize in 2005 for their work on AMS sketches.

Foundations

While the AMS sketches were initially introduced to compute the second frequency moment, since the reader might be more familiar with database terminology, the problem of estimating the size of join of two relations will be considered here instead. Notice that the size of the self join size of a relation coincides with the second frequency moment of the relation thus the treatment here is slightly more general but not more complicated.

Problem Setup

To set up the problem, assume access is provided to two relations F and G each with a single attribute a. Since it is convenient, denote with f_i and g_i the frequency of value i of attribute a in relation F and G, respectively. Assume that elements of F and G are streamed the result of the query: $\texttt{COUNT}(F \bowtie_a G)$ needs to be computed or estimated. Consider the following example:

Stream F:

a	1	1	2	3	1	3

frequency vector $\mathbf{f}$:

i	1	2	3
f_i	3	1	2

Stream G:

a	3	1	3	1	1

frequency vector $\mathbf{g}$:

i	1	2	3
g_i	3	0	2

Elements of F and G are assumed to arrive one by one (i.e., are streamed). If the frequency vectors $\mathbf{f}$ and $\mathbf{g}$ can be maintained, than the result of the query $\texttt{COUNT}(F \bowtie {}_aG)$ can be computed clearly in the following example:

$$\begin{aligned}\text{COUNT}(F \bowtie_\alpha G) &= \mathbf{fg}^T \\ &= [3 \quad 1 \quad 2][3 \quad 0 \quad 2]^T \\ &= 3 \cdot 3 + 1 \cdot 0 + 2 \cdot 2 \\ &= 13\end{aligned}$$

Observe that the size of join can be written as the dot product of the frequency vectors of the two relations. Expressing the size of the join in terms of frequencies of the join attribute is key for AMS sketch based approximation.

Main Idea

Assume now that the estimate $\texttt{COUNT}(F \bowtie {}_aG)$ needs to be computed but only less than linear space, in terms of the size of the frequency vectors, is available. As it turns out, exact computation is not possible with less space (in an asymptotic sense), but approximate computation is possible. The AMS sketches prove that they allow the approximation of the size of join using sub-linear space.

The main idea behind AMS sketches is to summarize the entire frequency table by projecting it on a random vector. The value thus obtained will be referred to as an elementary sketch. Then, use the two elementary sketches, one for each relation, to recover approximately the result of the query. Interestingly, a random vector $\xi = [\xi_1 \ldots \xi_n]$ of ± 1 values suffices to obtain projections with the desired properties. For simplicity, random vectors for which $\forall i$, $E[\xi_i] = 0$ are preferred. With this:

Sketch of F, $X_F = \mathrm{f}\xi^T$

- Sketch of G, $X_G = \mathrm{g}\xi^T$
- $X = X_F X_G$ estimates $\texttt{COUNT}(F \bowtie_a G)$ since

$$E[X] = E[\mathbf{f}\xi^T\xi\mathbf{g}^T] = \mathbf{f}E[\xi^T\xi]\mathbf{g}^T = \mathbf{f}I\mathbf{g}^T = \mathbf{fg}^T$$

if $E[\xi^T\xi] = I$. To ensure this, property distinct elements of ξ must be pair-wise independent, i.e., $\forall i \neq i',\ \xi_i^2 = 1,\ E[\xi_i\xi_{i'}] = 0$

For the particular random vector $\xi = [\xi_1\ \ \xi_2\ \ \xi_3] = [-1 + 1 - 1]$, the value of the elementary sketches and the overall estimate will be:

$$X_F = \mathbf{f}\xi^T = -4$$
$$X_G = \mathbf{g}\xi^T = -5$$
$$X = X_F X_G = (-4)(-5) = 20 \approx 13$$

The error of the estimate X is due to its variance that can be shown to have the property:

$$\mathrm{Var(X)} \leq 2\mathbf{ff}^{\mathrm{T}}\mathbf{gg}^{\mathrm{T}} = 2\ \mathrm{SJ(F)}\ \mathrm{SJ(G)}$$

as long as the random vector ξ is 4-wise independent, i.e., $\forall i_1 \neq i_2 \neq i_3 \neq i_4,\ E[\xi_{i_1}\xi_{i_2}] = 0,\ E[\xi_{i_1}\xi_{i_2}\xi_{i_3}\xi_{i_4}] = 0$

Since a higher degree of independence would not make the sketch more precise, 4-wise independence suffices. This is important since 4-wise independent ± 1 random vectors can be generated *on the fly* by combining a small seed s and the index of the entry using $\xi_i(s) = h(s,i)$ with h a special hash function that guarantees the 4-wise independence of the components of ξ. The fact that elements of ξ can be generated on the fly is important since space can be saved and, more importantly, because sketches X_F and X_G can be computed using constant storage. This is how this can be accomplished using the previous example:

$$\begin{aligned} X_F = \mathbf{f}\xi^T &= \sum_i f_i\xi_i \\ &= \sum_{t\in F}\xi_{t.a} = \xi_1 + \xi_1 + \xi_2 + \xi_3 + \xi_1 + \xi_3 \\ &= h(s,1) + h(s,1) + h(s,2) + \\ &\quad h(s,3) + h(s,1) + h(s,3) \end{aligned}$$

$$\begin{aligned} X_G = \mathbf{g}\xi^T &= \sum_i g_i\xi_i \\ &= \sum_{t\in G}\xi_{t.a} = \xi_3 + \xi_1 + \xi_3 + \xi_1 + \xi_1 \\ &= h(s,3) + h(s,1) + h(s,3) + h(s,1) + h(s,1) \end{aligned}$$

From this example, it can be observed that, to maintain the elementary sketches over the streams F and G, the only operation needed is to increment X_F and X_G by the value of $\xi_{t.a}$ using the function $h(\cdot)$ and the seed s where $t.a$ is the value of attribute a of the current tuple t arriving on the data stream. The fact that the elementary sketches can be computed so easily by considering one element at the time in an arbitrary order is what makes the AMS sketches appealing as an approximation technique.

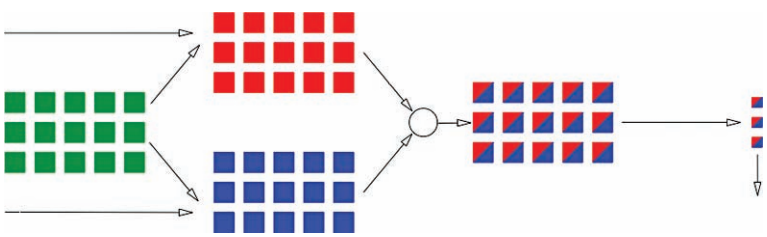
Improving the Basic Schema

Since the streams F and G are summarized by a single number X_F and X_G, respectively, it is not expected that the estimate will be very precise (this is suggested as well by the above example). In order to improve the accuracy of X, a standard technique in randomized algorithms can be used (i.e., generating multiple independent copies of random variable X). Copies of X are averaged in order to decrease the variance (thus the error). The median of such averaged values of X is used to estimate $\mathtt{COUNT}(F \bowtie_a G)$ since medians improve confidence. Multiple copies of X can be obtained using multiple seeds as depicted in Fig. 1.

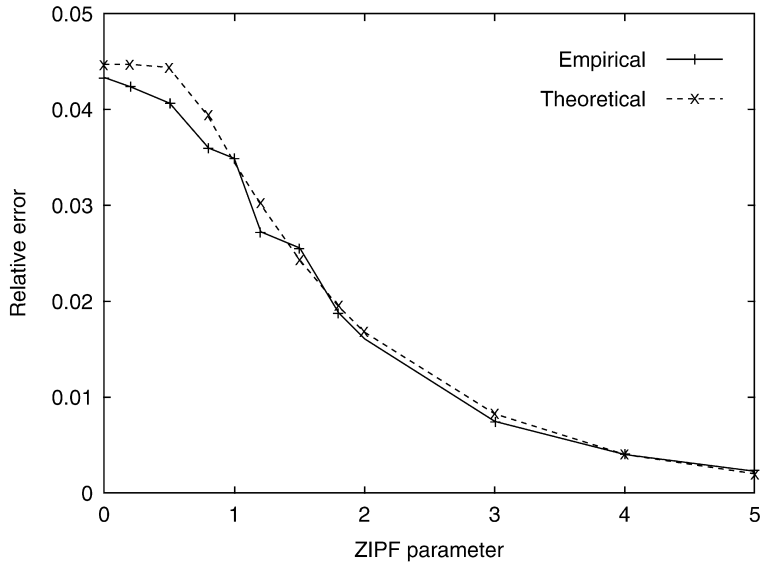
It can be shown that:

Average $\frac{8\mathrm{Var(X)}}{\epsilon^2 E^2[X]}$ independent copies of X to reduce error to $\in$

- Median of 2 log $1/\delta$ such averages increases the confidence to $1 - \delta$



AMS Sketch. Figure 1. Combining elementary sketches to estimate $\mathtt{COUNT}(F \bowtie_a G)$ with relative error at most $\in$ with probability at least $1 - \delta$.



AMS Sketch. Figure 2. Relative error of AMS sketches as a function of Zipf coefficient.

Key Applications

AMS sketches are particularly well suited for computing aggregates when data is either streamed (or a single pass over the data is allowed/desirable) or distributed at multiple sites. Thus, AMS sketches are relevant for processing large amount of data, as is the case in data warehousing, or processing distributed/streaming data, as is the case for computing networking statistics.

Experimental Results

To get an understanding of how the AMS sketches perform in the problem of estimating the self join size of a relation, consider the following setup. The domain of the attribute on which the self join size is computed is set to 16,384. The seize of the relation is fixed at 100,000 tuples. The distribution of the frequencies of join attribute values are generated according to a Zipf distribution with a varying Zipf coefficient. The number of medians is set to 1 (no median computation) and the number of elementary sketches averaged is set to 1,024.

The relative error, both theoretical and empirical, of AMS sketches is depicted in Fig. 2. The following observations confirm the intuition based on theory for the behavior of AMS sketches: (i) on the self join size problem the error is acceptable for sketches of size in the order of 2,000 words, (ii) the error decreases somewhat as the skew increases, and (iii) the theoretical prediction follows entirely the empirical behavior.

URL to Code

http://www.cs.rutgers.edu/~muthu/massdal-code-index.html http://www.cise.ufl.edu/~adobra/AQP/code.html

Cross-references

- ▶ Approximate Query Processing
- ▶ Data Stream

Recommended Reading

1. Alon N., Gibbons P.B., Matias Y., and Szegedy M. Tracking join and self-join sizes in limited storage. J. Comput. Syst. Sci., 64 (3):719–747, 2002.
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. Charikar M., Chen K., and Farach-Colton M. Finding frequent items in data streams. In Proc. 29th Int. Colloquium on Automata, Languages and Programming, 2002, pp. 693–703.
4. Cormode G. and Garofalakis M. Sketching streams through the net: distributed approximate query tracking. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 13–24.
5. 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.
6. Dobra A., Garofalakis M., Gehrke J., and Rastogi R. Processing complex aggregate queries over data streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 61–72.
7. Rusu F. and Dobra A. Pseudo-random number generation for sketch-based estimations. ACM Trans. Database Syst., 32(2):11, 2007.
8. Rusu F. and Dobra A. Statistical Analysis of Sketch Estimators. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 187–198.

Analogy

► Visual Metaphor

Anchor

► Anchor Text

Anchor Text

Vassilis Plachouras
Yahoo! Research, Barcelona, Spain

Synonyms

Anchor; Anchor text surrogate

Definition

Anchor text is the text associated with a hyperlink pointing from one Web document to another. Anchor text provides a concise description of a document, not necessarily written by the author of the document. It is very effective in Web search retrieval tasks, and in particular, in tasks where the aim is to find the home page of a given website.

Key Points

Hyperlink analysis algorithms explicitly employ the hyperlinks between Web documents to find high quality or authoritative Web documents. A form of implicit use of the hyperlinks in combination with content analysis is the use anchor text associated with the incoming hyperlinks of documents. Web documents can be represented by an anchor text surrogate, which is formed by collecting the anchor text associated with the hyperlinks pointing to the document.

The anchor text of the incoming hyperlinks provides a concise description for a Web document. The used terms in the anchor text may be different from the ones which occur in the document itself, because the author of the anchor text is not necessarily the author of the document. Eiron and McCurley [2] found similarities in the distribution of terms between the anchor text of Web documents and the queries submitted to an intranet search engine by users. Similarities were also found in the use of abbreviations and technical terms.

Craswell et al. [1] show that anchor text is effective for navigational search tasks and more specifically for finding home pages of Web sites. They report that searching for home pages of websites using an index of anchor text performs better than using an index of the document contents. Upstill et al. [3] also suggest that the anchor text of the incoming hyperlinks from documents outside a corpus of documents enhances the retrieval effectiveness for homepage finding.

Cross-references

► Document Links and Hyperlinks
► Field-Based Information Retrieval Models

Recommended Reading

1. Craswell N., Hawking D., and Robertson S. Effective site finding using link anchor information. In Proc. 24th Annu. Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001, pp. 250–257.
2. Eiron N. and McCurley K.S. Analysis of anchor text for web search. In Proc. 26th Annu. Int. ACM SIGIR Conf. on Research and Development in Informaion Retrieval, 2003, pp. 459–460.
3. Upstill T., Craswell N., and Hawking D. Query-independent evidence in home page finding. ACM Trans. Inform. Syst., 21(3):286–313, 2003.

Anchor Text Surrogate

► Anchor Text

AND-Join

► Workflow Join

AND-Split

► Split

Animation

► Dynamic Graphics

Annotation

Amarnath Gupta
University of California, San Diego, La Jolla, CA, USA

Definition

An annotation is any form of additional information "superposed" on any existing data or document.

Example: If a scientist records her experimental data in a relational database and then marks some "cells" of a table with the comment "consistent with previous findings," this additionally "marked" information is an annotation.

Key Points

Often annotations are not originally intended to be part of the collected data, and hence no data or schema structure was designed to hold it. Annotating data is a very common practice in science, where scientists would literally "mark" experimental observation with comments, and often use annotations to share their opinions in a collaborative study. As larger scale experiments are conducted and larger collaborations are formed, management of the annotated data becomes a serious challenge. In recent times, the emerging importance of annotation in scientific data management has been recognized by the Information Management community, leading to a variety of research in annotation management.

Cross-references

► Annotation-Based Image Retrieval
► Biomedical Scientific Textual Data Types and Processing
► Provenance

Recommended Reading

1. Bhagwat D., Chiticariu L., Tan W.C., and Vijayvargiya G. An annotation management system for relational databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 900–911.
2. Buneman P., Khanna S., and Tan W.-C. On propagation of deletions and annotations through views. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 150–158.
3. Geerts F., Kementsiesidis A., and Milano D. MONDRIAN: annotating and querying databases through colors and blocks. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 82.
4. Gertz M. and Sattler K.-U. Integrating scientific data through external, concept-based annotations. In Proc. Workshop on Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web, LNCS, Vol. 2590, Springer, 2002, pp. 220–240.
5. Murthy S., Maier D., and Delcambre L.M.L. Querying bi-level information. In Proc. 7th Int. Workshop on the World Wide Web and Databases, 2004, pp. 7–12.
6. Srivastava D. and Velegrakis Y. Intensional associations between data and metadata. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 401–412.

Annotation-based Image Retrieval

Xin-Jing Wang, Lei Zhang
Microsoft Research Asia, Beijing, China

Synonyms

Semantic image retrieval; Text-based image retrieval; Tag-based image search; Tag-based image retrieval

Definition

Given (i) a textual query, and (ii) a set of images and their annotations (phrases or keywords), annotation-based image retrieval systems retrieve images according to the matching score of the query and the corresponding annotations. There are three levels of queries according to Eakins [7]:

- Level 1: Retrieval by primitive features such as color, texture, shape or the spatial location of image elements, typically querying by an example, i.e., "find pictures like this."
- Level 2: Retrieval by derived features, with some degree of logical inference. For example, "find a picture of a flower."
- Level 3: Retrieval by abstract attributes, involving a significant amount of high-level reasoning about the purpose of the objects or scenes depicted. This includes retrieval of named events, of pictures with emotional or religious significance, etc., e.g., "find pictures of a joyful crowd."

Together, levels 2 and 3 are referred to as semantic image retrieval, which can also be regarded as annotation-based image retrieval.

Historical BackGround

There are two frameworks of image retrieval [6]: annotation-based (or more popularly, text-based) and content-based. The annotation-based approach can be

tracked back to the 1970s. In such systems, the images are manually annotated by text descriptors, which are used by a database management system (DBMS) to perform image retrieval. There are two disadvantages with this approach. The first is that a considerable level of human labor is required for manual annotation. The second is that because of the subjectivity of human perception, the manually labeled annotations may not converge. To overcome the above disadvantages, content-based image retrieval (CBIR) was introduced in the early 1980s. In CBIR, images are indexed by their visual content, such as color, texture, shapes. In the past decade, several commercial products and experimental prototype systems were developed, such as QBIC, Photobook, Virage, VisualSEEK, Netra, SIMPLIcity. Comprehensive surveys in CBIR can be found in [7,8].

However, the discrepancy between the limited descriptive power of low-level image features and the richness of user semantics, which is referred to as the "semantic gap" bounds the performance of CBIR. On the other hand, due to the explosive growth of visual data (both online and offline) and the phenomenal success in Web search, there has been increasing expectation for image search technologies. Because of these reasons, the main challenge of image retrieval is understanding media by bridging the semantic gap between the bit stream and the visual content interpretation by humans [3]. Hence, the focus is on automatic image annotation techniques.

Foundations

The state-of-the-art image auto-annotation techniques include four main categories [3,6]: (i) using machine learning tools to map low-level features to concepts, (ii) exploring the relations among image content and the textual terms in the associated metadata, (iii) generating semantic template (ST) to support high-level image retrieval, (iv) making use of both the visual content of images and the textual information obtained from the Web to learn the annotations.

Machine Learning Approaches

A typical approach is using Support Vector Machine (SVM) as a discriminative classifier over image low-level features. Though straightforward, it has been shown effective in detecting a number of visual concepts.

Recently there is a surge of interest in leveraging and handling relational data, e.g., images and their surrounding texts. Blei et al. [1] extends the Latent Dirichlet Allocation (LDA) model to the mix of words and images and proposed a Correlation LDA model. This model assumes that there is a hidden layer of topics, which are a set of latent factors and obey the Dirichlet distribution, and words and regions are conditionally independent on the topics, i.e., generated by the topics. This work used 7,000 Corel photos and a vocabulary of 168 words for annotation.

Relation Exploring Approaches

Another notable direction for annotating image visual content is exploring the relations among image content and the textual terms in the associated metadata. Such metadata are abundant, but are often incomplete and noisy. By exploring the co-occurrence relations among the images and the words, the initial labels may be filtered and propagated from initial labeled images to additional relevant ones in the same collection [3].

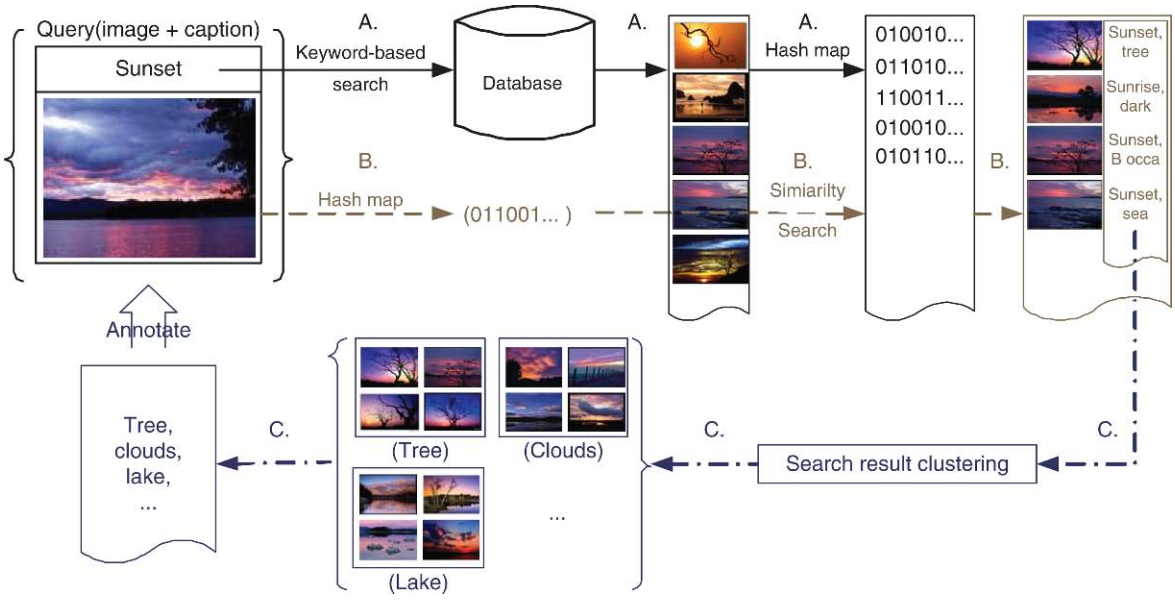
Jeon et al. [5] proposed a cross-media relevance model to learn the joint probabilistic distributions of the words and the visual tokens in each image, which are then used to estimate the likelihood of detecting a specific semantic concept in a new image.

Semantic Template Approaches

Though it is not yet widely used in the above mentioned techniques, Semantic Template (ST) is a promising approach in annotation-based image retrieval (a map between high-level concept and low-level visual features).

Chang and Chen [2] show a typical example of ST, in which a visual template is a set of icons or example scenes/objects denoting a personalized view of concepts such as meetings, sunset. The generation of a ST is based on user definition. For a concept, the objects, their spatial and temporal constraints, and the weights of each feature of each object are specified. This initial query scenario is provided to the system, and then through the interaction with users, the system finally converges to a small set of exemplar queries that "best" match (maximize the recall) the concept in the user's mind.

In contrast, Zhuang et al. [10] generates ST automatically in the process of Relevance Feedback, whose basic idea is to refine retrieval outputs based on interactions with the user. A semantic lexicon called WordNet is used in this system to construct a network of ST. During the retrieval process, once the user submits a query concept (keyword), the system can find a corresponding ST, and thus target similar images.



Annotation-based Image Retrieval. Figure 1. Framework of the search-based annotation system.

Large-Scale Web Data Supported Approaches

Good scalability to a large set of concepts is required in ensuring the practicability of image annotation. On the other hand, images from the Web repositories, e.g., Web search engines or photo sharing sites, come with free but less reliable labels. In [9], a novel search-based annotation framework was proposed to explore such Web-based resources. Fundamentally, it is to automatically expand the text labels of an image of interest, using its initial keyword and image content.

The process of [9] is shown in Fig. 1. It contains three stages: the text-based search stage, the content-based search stage, and the annotation learning stage, which are differentiated using different colors (black, brown, blue) and labels (A., B., C.). When a user submits a query image as well as a query keyword, the system first uses the keyword to search a large-scale Web image database (2.4 million images crawled from several Web photo forums), in which images are associated with meaningful but noisy descriptions, as tagged by "A." in Fig. 1. The intention of this step is to select a semantically relevant image subset from the original pool. Visual feature-based search is then applied to further filter the subset and save only those visually similar images (the path labeled by "B." in Fig. 1). By these means, a group of image search results which are both semantically and visually similar to the query image are obtained. To speed-up the visual feature-based search procedure, a hash encoding algorithm is adopted to map the visual features into hash codes, by which inverted indexing technique in text retrieval area can be applied for fast retrieval. At last, based on the search results, the system collects their associated textual descriptions and applies the Search Result Clustering (SRC) algorithm to group the images into clusters. The reason of using SRC algorithm is that (i) it is proved to be significantly effective in grouping documents semantically; and (ii) more attractively, it is capable of learning a name for each cluster that best represents the common topics of a clusters member documents. By ranking these clusters according to a ranking function, and setting a certain threshold, the system selects a group of clusters and merges their names as the final learnt annotations for the query image, which ends the entire process (C. in Fig. 1).

Key Applications

Due to the explosive growth of visual data (both online and offline), effective annotation-based image search becomes a highly-expected technique which facilitate human's lives.

Cross-references

- ▶ Cross-Modal Multimedia Information Retrieval
- ▶ Hash-Based Indexing
- ▶ Image Querying
- ▶ Image Retrieval
- ▶ Indexing and Similarity Search

- ▶ Information Information Retrieval
- ▶ Object Recognition
- ▶ Multimedia Information Retrieval
- ▶ Web Search and Crawling

Recommended Reading

1. Blei D. and Jordan M.I. Modeling Annotated Data. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 127–134.
2. Chang S.-F., Chen W., and Sundaram H. Semantic Visual Templates: Linking Visual Features to Semantics. In Proc. Int. Conf. on Image Processing, Vol. 3. 1998, pp. 531–534.
3. Chang S.-F., Ma W.-Y., and Smeulders A. Recent Advances and Challenges of Semantic Image/Video Search. In Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing, 2007, pp. 1205–1208.
4. Eakins J. and Graham M. Content-based image retrieval, Technical Report, University of Northumbria at Newcastle, 1999.
5. Jeon J., Lavrenko V., and Manmatha R. Automatic Image Annotation and Retrieval Using Cross-Media Relevance Models, In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 119–126.
6. Liu Y., Zhang D., Lu G., and Ma W.-Y. A survey of content-based image retrieval with high-level semantics. Pattern Recognit., 40(1):262 –282, 2007.
7. Long F., Zhang H.J., and Feng D.D. Fundamentals of content-based image retrieval. In Multimedia Information Retrieval and Management, D. Feng (eds.). Springer, 2003.
8. Rui Y., Huang T.S., and Chang S.-F. Image retrieval: current techniques, promising directions, and open issues, J. Visual Commun. Image Represent. 10(4):39–62, 1999.
9. Wang X.-J., Zhang L., Jing F., and Ma W.-Y. AnnoSearch: Image Auto-Annotation by Search. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition, 2006, pp. 1483–1490.
10. Zhuang Y., Liu X., and Pan Y. Apply Semantic Template to Support Content-based Image Retrieval. In Proc. SPIE, Storage and Retrieval for Media Databases, vol. 3972, December 1999, pp. 442–449.

Anomaly Detection on Streams

Spiros Papadimitriou
IBM T.J. Watson Research Center, Hawthorne, NY, USA

Definition

Anomaly detection generally refers to the process of automatically detecting events or behaviors which deviate from those considered normal. It is an unsupervised process, and can thus detect anomalies which have not been previously encountered. It is based on estimating a model of typical behavior from past observations and consequently comparing current observations against this model. It can be performed either on a single stream or among multiple streams. Anomaly detection encompasses outlier detection as well as change detection and therefore is closely related to forecasting and clustering methods.

Historical Background

Anomaly detection in streams has close connections to traditional outlier detection, as well as to change detection. The former is a common and widely studied topic in statistics [11]. The latter emerged in the context of statistical monitoring and control for continuous processes and the widely used CUSUM algorithm was proposed as early as 1954 [9]. With the emergence of data stream management systems, anomaly detection in this setting has received significant attention, with applications in network management and intrusion detection, environmental monitoring, and surveillance, to mention a few.

Foundations

Anomaly detection is closely related to outlier detection and change detection. After a review of the main ideas, the streaming case is presented.

Outlier Detection

The existing approaches to outlier detection can be broadly classified into the following categories. Typically, outlier detection relies on a model for the data. Model parameters are estimated based on appropriately chosen historical data. As new observations arrive, they are either compared directly against the model and are declared outliers if the fit is poor. Alternatively, a second set of model parameters may be estimated from recent observations. If there is a statistically significant difference among the two sets of parameters, the new observations are declared as outliers.

Clustering-Based and Forecasting-Based Approaches

Many clustering and forecasting algorithms detect outliers as by-products. However, not all clustering or forecasting procedures can be easily turned into outlier detection procedures.

Distribution-Based Approaches Methods in this category are typically found in statistics textbooks. They deploy some standard distribution model (e.g., Gaussian) and flag as outliers those objects which deviate from the model. These work well in many occasions, but may be unsuitable for high-dimensional data sets, or when

reasonable assumptions about the distribution of data points cannot be made.

Distance-Based and Density-Based Approaches A point in a data set is a distance-based outlier if at least a fraction β of all other points are further than r from it. This outlier definition is based on a single, global criterion determined by the parameters r and β. This can lead to problems when the data set has both dense and sparse regions. Density-based approaches aim to remedy this problem, by relying on the local density of each point's neighborhood.

Change Detection

Sequential hypothesis testing and sequential change detection arose out of problems in statistical process control. Assume a collected sequence of observations, modeled as random variables $X_1, X_2, ..., X_t, ...$. Additionally, assume that X_t are drawn from a distribution with parameter θ and that a test of whether the true parameter is θ_0 or θ_1 is desired.

The Sequential Likelihood Ratio Test (SLRT) relies on the logarithm of likelihood ratios $z_t := \log(p(x_t; \theta_0)/p(x_t; \theta_1))$ and tests the cumulative sum $z_1 + ... + z_t$ to decide upon the true parameter.

This can be extended to other settings, such as detecting changes in other distribution parameters. For example, in its simplest form, CUSUM tests for a shift in the mean by essentially applying SLRT, assuming points independently drawn from a Gaussian distribution with known variance. Many other versions have appeared since the CUSUM test was first proposed [9], relaxing or modifying some of these assumptions.

In general, change detection is closely related to outlier detection; in fact, change detection may also be viewed as outlier detection along the time axis.

Streaming Algorithms

In a streaming setting, there are two key challenges that need to be addressed:

1. *Limited resources.* In a streaming setting, a large number of observations arrives over time and the total volume of data grows indefinitely. However processing and storage capacity are limited, in comparison to the amount of data. Therefore, data summarization or *sketching* techniques need to be applied, in order to extract a few, relevant features from the raw data.
2. *Concept drift.* In an indefinitely growing collection of observations, changes in the underlying features (e.g., distribution parameters) may not necessarily correspond to anomalies, but rather be part of normal changes in the behavior of the system. Thus, mechanisms to handle such non-stationarity or *concept drift* and adapt to changing behavior are necessary [12].

Next, several of the approaches that have been studied in the literature are reviewed.

Sketching techniques In the past several years, a number of techniques for *sketch* or *synopsis* construction have appeared, with applications to many stream processing problems. Some examples include CM sketches, AMS sketches, FM sketches, and Bloom filters [2]. Other summarization techniques specifically for data clustering on streams have appeared, such as those in [1,4], which can be easily extended for outlier detection on streams.

Burst Detection In many applications, the appearance of sudden bursts in the data often signifies an anomaly. For example, in a network monitoring application, a burst in the traffic volume to a particular destination may signify a denial of service (DoS) attack. Thus, *burst detection* on streams has received significant attention. Examples of such work include [7] and [14].

Correlation Dnalysis Often a collection of multiple streams is available and measurements from different streams may be highly correlated with each other. If the strength of correlations changes over time [13] or the number of correlated components varies [10], this often signifies changes in the underlying data-generating process that may be due to anomalies.

Change Analysis More generally, detecting significant changes has been studied in the context of stream processing [3].

Key Applications

Intrusion Detection With the widespread adoption of the internet, various forms of malware (e.g., viruses, worms, trojans, and botnets) have become a serious and costly issue. Most intrusion detection systems (IDS) rely on known signatures to identify malicious payloads or behaviors. However, there are several efforts underway

for automatic detection of suspicious activity on the fly, as well as for automating the signature extraction process.

System Monitoring Maintenance costs for large computer clusters or networks is traditionally labor-intensive and contributes a large fraction of total cost of ownership. Hence, autonomic computing initiatives aim at automating this process. An important first step is the automatic, unsupervised detection of abnormal events (e.g., node or link failures) based on continuously collected system metrics. Streaming anomaly detection methods are used to address this problem.

Process Control Applications in quality control and industrial process control have traditionally provided much of the impetus for the development of change detection methods. Machinery used in a production chain (e.g., food preparation or chip fabrication) typically monitor a large number of process parameters at each step. Early detection of sudden changes in those parameters is important to identify potential flaws in the process which can severely affect end product quality.

Pervasive Healthcare Small and cheap sensors which can continuously monitor patient physiological data (e.g., temperature, blood pressure, heart rate, ECG measurements, glucose levels, etc.) are becoming widely available. Anomaly detection methods can prove essential in enabling early diagnosis of potential life-threatening conditions, as well as preventive healthcare.

Civil Infrastructure Early detection of faults by continuously monitoring civil infrastructure components (e.g., bridges, buildings, and roadways) can reduce maintenance costs and increase safety. Similarly, surveillance systems on urban environments rely on anomaly detection methods to spot suspicious activities and increase security.

Future Directions

Certain anomalies can be detected only by taking into account information collected from a large number of different sources. Even if data ownership issues are resolved, collecting all this information at a central site is often infeasible due to its large volume. A number of efforts have tackled this problem in the past few years, but much remains to be done, especially as the scale of information collected increases. Also related to this trend is anomaly detection on more complex data, such as time-evolving graphs.

Cross-references

- ▶ Change Detection
- ▶ Clustering
- ▶ Forecasting
- ▶ Outlier Detection

Recommended Reading

1. 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.
2. Aggarwal C.C. and Yu P.S. A survey of synopsis construction in data streams. In Data Streams: Models and Algorithms. Springer, 2007.
3. Cormode G. and Muthukrishnan S. What's new: finding significant differences in network data streams. IEEE/ACM Trans. Netw., 13(6):1219–1232, 2005.
4. Guha S., Meyerson A., Mishra N., Motwani R., and O'Callaghan L. Clustering data streams: theory and practice. IEEE Trans. Knowl. Data Eng., 15(3):515–528, 2003.
5. 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, pp. 97–106.
6. Jain A.K., Narasimha Murty M. and Flynn P.J. Data clustering: a review. ACM Comput. Surv., 31(3):264–323, 1999.
7. Kleinberg J. Bursty and hierarchical structure in streams. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 91–101.
8. Lee W., Stolfo S.J., and Mok K.W. Adaptive intrusion detection: a data mining approach. Artif. Intell. Rev. 14(6):533–567, 2000.
9. Page E.S. Continuous inspection schemes. Biometrika, 41(1):100–115, 1954.
10. Papadimitriou S., Sun J., and Faloutsos C. Streaming pattern discovery in multiple time-series. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 697–708.
11. Peter J.R. and Annick M.L. Robust Regression and Outlier Detection. Wiley, New York, 1987.
12. 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.
13. Zhu Y. and Shasha D. StatStream: statistical monitoring of thousands of data streams in real time. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 358–369.
14. Zhu Y. and Shasha D. Efficient elastic burst detection in data streams. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 336–345.

Anonymity

Simone Fischer-Hübner
Karlstad University, Karlstad, Sweden

Synonyms

Namelessness; Nonidentifiability

Definition

The term anonymity originates from the Greek word "anonymia," which means "without a name."

In the context of computing, anonymity has been defined in [2] as follows: "Anonymity of a subject means that the subject is not identifiable within a set of subjects, the anonymity set." The anonymity set is the set of all possible subjects, e.g., the set of all possible senders of a message or the set of all possible recipients of a message (dependent on the knowledge of an attacker). *Sender anonymity* means that a message cannot be linked to the sender, while *receiver anonymity* implies that a certain message cannot be linked to the receiver of that message. Relationship anonymity of a sender and recipient means that even though a sender and a recipient can be identified as participating in some communication, they cannot be identified as communicating with each other, i.e., sender and recipient are unlinkable. The definition above corresponds to the definition of anonymity in [1]: "Anonymity of a user means that the user may use a resource or service without disclosing the user's identity."

To reflect the possibility to quantify anonymity, a slightly modified definition has also provided by [2]: "Anonymity of a subject from the attacker's perspective means that the attacker cannot sufficiently identify the subject within a set of subjects, the anonymity set."

Data protection legislation usually defines data as anonymous if information concerning personal or material circumstances can no longer (or with disappropriate amount of time, expense, and labor) be attributed to an individual.

Key Points

Providing anonymity is the best strategy for achieving privacy. If individuals can act anonymously and if their data are kept in an anonymous form, their privacy is not affected (and consequently, data protection legislation is not applicable).

For providing anonymity, however, it has to be guaranteed that potential attackers cannot or not sufficiently identify the individuals. Proposals for measuring the degree of anonymity are usually based on Shannon's entropy.

Cross-references

- ▶ Privacy
- ▶ Privacy-Enhancing Technologies
- ▶ Privacy Metrics

Recommended Reading

1. Common Criteria Project, Common criteria for information technology security evaluation, Version 3.1, Part 2: Security functional requirements, September 2006, www.commoncriteriaportal.org
2. Pfitzmann A. and Hansen M. "Anonymity, unlinkability, unobservability, pseudonymity, and identity management – a consolidated proposal for terminology," Version 0.29, http://dud.inf.tu-dresden.de/Anon_Terminology.shtml, July, 2007.

Anonymity in Location-based Services

▶ Spatial Anonymity

ANSI/INCITS RBAC Standard

Yue Zhang, James B. D. Joshi
University of Pittsburgh, Pittsburgh, PA, USA

Synonyms

RBAC standard

Definition

The ANSI/INCITS RBAC standard includes the *core*, *hierarchical*, and the *constraint RBAC* models [1]. The *core RBAC* includes the following entities:

- *USERS*, *ROLES*, *OPS*, and *OBS* are the sets of users, roles, operations and objects, respectively.
- $UA \subseteq USERS \times ROLES$ is a many-to-many mapping from *USERS* to *ROLES*.
- *assigned_users*: $(r{:}ROLES) \rightarrow 2^{USERS}$, is mapping of role r onto a set of users. Formally: *assigned_users* $(r) = \{u \in USERS \mid (u, r) \in UA\}$
- $PRMS = 2^{(OBS \times OPS)}$ is the set of permissions.
- $PA \subseteq PERMS \times ROLES$ is a many-to-many mapping from *PERMISSIONS* to *ROLES*.
- *assigned_permissions*$(r{:}ROLES) \rightarrow 2^{PRMS}$, is mapping of role r onto a set of permissions. Formally: *assigned_permissions*$(r) = \{p \in PRMS \mid (p, r) \in PA\}$
- $Op(p{:}\ PRMS) \rightarrow \{op \subseteq OPS\}$, is the permission to operation mapping.
- $Ob(p{:}\ PRMS) \rightarrow \{ob \subseteq OBS\}$, is the permission to object mapping.
- *SESSIONS* = the set of sessions
- *session_users* $(s{:}SESSIONS) \rightarrow USERS$ is mapping of session s onto the corresponding user.

– *session_roles* (*s*:*SESSIONS*) → 2^{ROLES} is mapping of session *s* onto a set of roles.
– *avail_session_perms*(*s*:*SESSIONS*) → 2^{PRMS} gives the permissions available to a user in a session =

$$\left[\bigcup_{r \in session\_roles(s)} assigned\_permissions(r)\right]$$

The *hierarchical RBAC* model extends the core RBAC model with the *general* and *restricted* role hierarchies. The *general role hierarchy* is defined below:

- $RH \subseteq ROLES \times ROLES$ is a partial order on *ROLES*, written as $\geq$, where $r_1 \geq r_2$ only if all permissions of r_2 are also permissions of r_1, and all users of r_1 are also users of r_2.
- *authorized_users*(*r*: *ROLES*) → 2^{USERS}, is mapping of role *r* onto a set of users in the presence of a role hierarchy. Formally: $authorized\_users(r) = \{u \in USERS \mid r' \geq r, (u, r') \in UA\}$.
- *authorized_permissions*(*r*: *ROLES*) → 2^{PRMS}, is mapping of role *r* onto a set of permissions in the presence of a role hierarchy. Formally: $authorized\_permissions(r) = \{p \in PRMS \mid r' \geq r, (p, r') \in PA\}$

The *restricted hierarchy* is similar to the *general role hierarchies* with the following limitation:

$$\forall r, r_1, r_2 \in ROLES, \quad r \geq r_1 \wedge r \geq r_2 \Rightarrow r_1 = r_2$$

The *constraint RBAC* model extends the hierarchical RBAC with *Static Separation of Duty* (SSoD) and *Dynamic SoD* (DSoD) constraints.

SSoD Constraint: $SSD \subseteq (2^{ROLES} \times N)$ is a collection of pairs (*rs*, *n*) in *Static Separation of Duty*, where each *rs* is a role set, and *n* is a natural number ≥ 2, with the property that no user is assigned to *n* or more roles from the set *rs* in each (*rs*, *n*) *SSD*.

DSoD Constraint: $DSD \subseteq (2^{ROLES} \times N)$ is collection of pairs (*rs*, *n*) in *Dynamic Separation of Duty*, where each *rs* is a role set and *n* is a natural number ≥ 2, with the property that no subject may activate *n* or more roles from the set *rs* in each *dsd DSD*.

Key Points

The ANSI/INCTIS RBAC standard essentially evolved from the original RBAC96 model [2]. Several advanced features of RBAC such as cardinality constraints, hybrid hierarchy, much fine-grained SoD constraints, features related to the administration of RBAC, etc., are not covered in the standard. The link (http://csrc.nist.gov/groups/SNS/rbac/standards.html) includes the most updated information on ANSI RBAC.

Cross-references

▶ Role Based Access Control

Recommended Reading

1. ANSI. American national standard for information technology – role based access control. ANSI INCITS, 359–2004, February 2004.
2. Sandhu R.S., Coyne E.J., Feinstein H.L., and Youman C.E. Role-based access control models. IEEE Comput., 29(2):38–47, 1996.

Answering Queries Using Views

Vasilis Vassalos
Athens University of Economics and Business, Athens, Greece

Synonyms

Query rewriting using views

Definition

Answering queries using views refers to a data management problem and the set of related techniques, algorithms and other results to address it. The basic formulation of the problem is the following: Given a query Q over a database schema Σ, expressed in a query language L_Q and a set of views $V_1, V_2, ..., V_n$ over the same schema, expressed in a query language L_V, is it possible to answer the query Q using (only) the views $V_1, V_2, ..., V_n$?

The problem has a number of related formulations: What is the *maximal set* of tuples in the answer of Q that can be obtained from the views? If it is possible to access both the views and the database relations, what is the cheapest query execution plan for answering Q?

From the above, it is clear that this is more generally a family of problems. Problems of different complexity, often admitting different (or no) solutions and solution techniques, result from making choices about various "parameters" of the original problem. For example, one can choose the languages L_Q and L_V, the language L_R used to answer Q from the views, the ability to access the relations of the schema Σ, the possible existence of constraints, e.g., equality-generating or tuple-generating

dependencies, as well as the semantics of the views (sound, complete, or both).

The problem is also referred to as query rewriting using views.

Historical Background

The problem of putting together information from multiple sources has a relatively long history within the database community. In the 1970s, distributed databases offered a controlled solution to the problem: the data are structured with a single schema and are put under the control of a single, albeit distributed, database system. In the early 1980s, the challenges of integrating full-fledged relational databases was studied in the context of multidatabases. An important direction of multidatabase research was static schema integration, i.e., creating in advance a single new schema with as much of the information of the original schemas as possible. Query processing could then be performed on the integrated schema. "Reusing" the original database tables also received some attention in the multidatabase context. In the early 1990s, techniques were developed in order to make use of existing materialized views to speed up the processing of queries that did not mention them explicitly. These techniques were precursors to the more general techniques developed a few years later for the problem of answering queries using views. The problem was cast in its current form in a seminal paper by Levy, Mendelzon, Sagiv, and Srivastava in 1995.

Foundations

Preliminaries

To fully specify the problem of answering queries using views one needs to decide on the languages used for the queries, the views and, in some cases, the rewritings. This entry presents the basic case of answering *conjunctive queries* using *conjunctive views*, possibly in the presence of constraints. The relevant definitions are presented next. Techniques developed for different languages or data models, such as bag queries (and views), XML queries, queries with aggregates, etc., can be found in the Recommended Reading list. Moreover, detailed presentations of the techniques discussed in the article, including exceptions, optimizations, and other technical issues, can be found in the research articles on the Recommended Reading list.

Conjunctive Queries A *conjunctive* query can be represented as:

$$q(\bar{X}) \leftarrow s_1(\bar{X}_1) \wedge \ldots \wedge s_n(\bar{X}_n)$$

where q and $s_1,\ldots,s_n$ are predicate names. In general, $s_1,\ldots,s_n$ refer to database relations. The atoms $s_1(\bar{X}_1),\ldots, s_n(\bar{X}_n)$ that appear in the body of the query are called *subgoals* of the query, The atom $q(\bar{X})$ is called the *head* of the query and defines the answer relation. The tuples $\bar{X}$, $\bar{X}_1,\ldots,\bar{X}_n$ contain either variables or constants. The variables in $\bar{X}$ are the *distinguished* variables of the query, the rest of the variables are the *existential* variables. An important condition for a conjunctive query is *safety*: A query is safe if $\bar{X} \subseteq \bar{X}_1 \cup \ldots \cup \bar{X}_n$, i.e., every distinguished variable must also appear in a query subgoal.

Use $Vars(Q)$ and $Subg(Q)$ to refer to the set of variables (and constants) in Q and subgoals of Q respectively. $Q(D)$ refers to the result of evaluating the query Q over the database D.

Views A *view* is a query whose head defines a new database relation. If this relation is not stored, the view is called a *virtual* view. If the results of executing the view are stored, it is called a *materialized* view and the relation is the *extension* of the view. Denote by D_V the database D extended with the extensions of the views belonging to a view set V.

Containment & Equivalence The concepts of query containment and equivalence are central to query rewriting theory [ref Encyclopedia article Query Rewriting] as they are used to test the correctness of a rewriting of a query using a set of views.

Definition 1 *A query Q_1 is contained in a query Q_2, denoted $Q_1 \sqsubseteq Q_2$, if for all database instances D, the set of tuples computed for Q_1 is a subset of those computed for Q_2, i.e., $Q_1(D) \subseteq Q_2(D)$. The two queries are equivalent if $Q_1 \sqsubseteq Q_2$ and $Q_2 \sqsubseteq Q_1$.*

Containment mappings provide a necessary and sufficient syntactic condition for testing query containment of conjunctive queries.

Definition 2 *A mapping τ from $Vars(Q_2)$ to $Vars(Q_1)$* is a *containment mapping if*

- *It is the identity on constants*
- *It maps every subgoal $inSubg(Q_2)$ to a subgoal $inSubg(Q_1)$, and*

- *It maps the head of Q_2 to the head of Q_1.*

A seminal result in semantic query optimization is that a query Q_1 is contained in Q_2 if and only if there is a containment mapping from Q_2 to Q_1.

Rewritings

Given a query Q and a set of views $V_1, V_2,...,V_n$ over the same database schema, the goal is to find an *equivalent rewriting* Q' of the query, i.e., a query Q' that uses one or more of the views in the body.

Definition 3 *A query Q' is an equivalent rewriting of query Q that uses the set of views $V = \{V_1, V_2,...,V_n\}$* if

- *Q and Q' are equivalent, and*
- *Q' refers to the views in V.*

For every database D, Q' is evaluated over D_V. A rewriting Q' is *locally minimal* if no literals from Q' can be removed while retaining equivalence to Q. A rewriting is *globally minimal* if there is no other rewriting with fewer literals. If the rewriting refers *only* to the views in V and to no other relations then the rewriting is called *complete*. When looking for rewritings, it is preferable to find those that are cheaper to evaluate than the original query. Below is an example of a query and a rewriting using views.

Example 1: Consider the following query Q and view V

$$Q : q(X, U) \leftarrow p(X, Y) \wedge p_0(Y, Z) \wedge p_1(X, W) \wedge p_2(W, U).$$

$$V : v(A, B) \leftarrow p(A, C) \wedge p_0(C, B) \wedge p_1(A, D).$$

Q can be rewritten using V as follows:

$$Q' : q(X, U) \leftarrow v(X, Z) \wedge p_1(X, W) \wedge p_2(W, U).$$

By substituting the view, the first two literals of the query can be removed. However, although the third literal in Q is guaranteed to be satisfied by V, it cannot be removed from the query, as the variable D is projected out in the head of V. Hence, if p_1 were removed from the query, the join condition between p_1 and p_2 could not be enforced.

In several settings, *maximally contained* rewritings need to be considered. Unlike equivalent rewritings, maximally contained rewritings may differ depending on the language used to express the rewritings. Therefore, the following definition depends on a particular query language:

Definition 3 *Let Q be a query and $V = \{V_1,...,V_m\}$ be a set of views over the same database schema, and L be a query language. The query Q' is a maximally contained rewriting of Q* using *V with respect to L* if:

- *Q' is a query in L that refers only to the views in V,*
- *$Q' \sqsubseteq Q$, and*
- *for every rewriting $Q_1 \in L$, such that $Q_1 \sqsubseteq Q$, $Q_1 \sqsubseteq Q'$.*

Characterizing Rewritings

Answering queries using views is closely related to the problem of query containment. The following proposition provides a necessary and sufficient condition for the existence of a rewriting of Q that includes a view V.

Proposition 1 *Let Q* and *V be conjunctive queries with built-in predicates. There is a rewriting of Q using V* if and only if *$\pi_\emptyset(Q) \sqsubseteq \pi_\emptyset(V)$, i.e., the projection of Q onto the empty set of columns is contained in the projection of V onto the empty set of columns.*

This proposition provides a complete characterization of the problem of using views for query answering. Two other important characteristics of the problem are that, for conjunctive queries and views, a rewriting that does not introduce new variables and does not include database relations that do not appear in the original query, can always be found. These characteristics allow significant pruning of the search space for a minimal rewriting of Q, but do not always hold for more expressive settings of the problem. Finally, a minimal rewriting of a query Q using a set of views V without built-in predicates does not need more than the number of literals in the query. In particular, *if the body of Q has p literals and Q' is a locally minimal and complete rewriting of Q using V, then Q' has at most p literals.*

The bound provided above does not hold when the database relations have functional dependencies. In such a case the size of a minimal rewriting is at most $p + d$ literals, where d is the sum of the arities of the literals in Q. In the presence of built-in predicates, the size of the rewritten query is at most exponential in the size of Q.

The above properties determine the complexity of the problem. In particular, if Q is a conjunctive query with built-in predicates and V is a set of conjunctive views *without* built-in predicates, then the problem of

determining whether there exists a rewriting of Q that uses V is NP-complete. If the views in V have built-in predicates, the problem is Π_2^p-complete. If neither the query nor the views have built-in predicates, then finding a rewriting with at most k literals, where k is the number of literals in the body of Q, is NP-complete. Finally, if the query and the views have built-in predicates, then finding a rewriting with at most k literals is in Σ_3^p.

Techniques for Answering Queries Using Views

The above characterization suggests a two-step algorithm for finding rewritings of a query Q using a set of views V. At first, find some containment mapping from V to Q and add to Q the appropriate atoms of V, resulting in a new query Q'. Then, minimize Q' by removing literals from Q that are redundant. An algorithm needs to consider every possible conjunction of k or fewer view atoms, where k is the number of subgoals in the query. Algorithms that attempt to explore more effectively the search space to produce maximally contained rewritings include the Bucket Algorithm, the Inverse-Rules Algorithm and the MiniCon Algorithm.

The Bucket Algorithm The main idea underlying the algorithm is that the query rewritings that need to be considered can be significantly reduced if for each subgoal in the query the relevant views are identified.

To demonstrate the algorithm, the following query and views are used (The example used is from [10]):

$$Q_1(x) \leftarrow cites(x, y) \wedge cites(y, x) \wedge sameTopic(x, y)$$
$$V_1(a) \leftarrow cites(a, b) \wedge cites(b, a)$$
$$V_2(c, d) \leftarrow sameTopic(c, d)$$
$$V_3(f, h) \leftarrow cites(f, g) \wedge cites(g, h) \wedge sameTopic(f, g)$$

Given a query Q, the Bucket Algorithm proceeds in two steps. In the first step, the algorithm creates a bucket for each subgoal in Q that is not in $C(Q)$, where by $C(Q)$ refers to the subgoals of comparison predicates of Q. Each bucket contains the views that are relevant to answering the particular subgoal. For the example, the following buckets are created:

cites(x, y)	*cites*(y, x)	*sameTopic*(x, y)
$V_1(x)$	$V_1(x)$	$V_2(x, y)$
$V_3(x, y)$	$V_3(x, y)$	$V_3(x, y)$

The algorithm also requires that every distinguished variable in the query should be mapped to a distinguished variable in the view. Hence, the algorithm does not include the entry $V_1(y)$ even though it is possible to map the subgoal $cites(x, y)$ in the query to the subgoal $cites(b, a)$ in V_1.

In the second step, for each element of the Cartesian product of the buckets, the algorithm constructs a conjunctive rewriting and checks whether it is contained (or can be made to be contained) in the query. If so, the rewriting is added to the answer. Therefore, the result of the Bucket Algorithm is a union of conjunctive rewritings.

In the example, the algorithm will try to combine V_1 with the other views and fail. Then it will consider the rewritings involving V_3 and V_2, and discover a contained rewriting. Finally, it will consider the rewritings involving only V_3, and find a contained rewriting by equating the variables in the head of V_3. It is possible to add an additional check that will determine whether a resulting rewriting will be redundant, as is the case here with the rewriting combining V_3 and V_2. Therefore, the only minimal rewriting in this example is: $Q_1'(x) \leftarrow V_3(x, x)$. This rewriting is in fact an equivalent one.

The Bucket Algorithm misses important interactions between view subgoals by considering each subgoal separately. Consequently, the buckets can contain unusable views and, as a result, the second step of the algorithm can become very expensive.

The Inverse Rules Algorithm The main idea of the Inverse-Rules Algorithm is to construct a set of rules that *invert* the view definitions, i.e., rules that compute tuples for the database relations from tuples of the views. The inverse rules together with a conjunctive query Q constitute a maximally contained rewriting for Q that is represented as a union of conjunctive queries. Considering the views of the previous example, the algorithm would construct the following inverse rules:

$$R_1 : cites(a, f_{V_1}(a)) \leftarrow V_1(a)$$
$$R_2 : cites(f_{V_1}(a), a) \leftarrow V_1(a)$$
$$R_3 : sameTopic(c, d)) \leftarrow V_2(c, d)$$
$$R_4 : cites(f, f_{V_3}(f, h)) \leftarrow V_3(f, h)$$
$$R_5 : cites(f_{V_3}(f, h), h) \leftarrow V_3(f, h)$$
$$R_6 : sameTopic(f, f_{V_3}(f, h)) \leftarrow V_3(f, h)$$

For every view relation V with an existential variable Z_i, a function symbol $f_{V,i}$ is introduced. The above rules provide all the information about the database relations that can be extracted from the view extension. Intuitively, a tuple of the form (A) in the extension of the view V_1 is a *witness* of two tuples in the relation *cites*: a tuple of the form (A, Z), for some value of Z, and a tuple of the form (Z, A), for the same value of Z. The rules will generate from a tuple $V_1(A)$ two such tuples that will contain the unique functional term $f_{V_1}(A)$: it is a syntactic stand-in for the unknown (but known to exist) common value in the two tuples.

The rewriting of a query Q using the set of views V is simply the query consisting of Q and the inverse rules for V (There is a systematic way to eliminate the function symbols). Two important advantages of the algorithm are that the inverse rules can be constructed ahead of time, independent of a particular query, in polynomial time, and that the technique is also applicable virtually without change to recursive queries.

The rewritings produced by the Inverse-Rules Algorithm have some shortcomings when used for query evaluation. First, applying the inverse rules to the extension of the views may invert some of the computation done to produce the extent of the view. Second, they may access views that have no relevance to the query.

The MiniCon Algorithm The MiniCon Algorithm starts by considering, for each subgoal in the query, which view specializations contain subgoals that can "cover" it. A view specialization is created from a view by possibly equating some head variables. Once the algorithm finds a partial mapping φ from a subgoal g in the query to a subgoal g_1 in the view specialization V, it extends it to a minimal additional set of subgoals G of the query that must also be mapped to V. In particular, G includes all subgoals of the query that contain some variable of g mapped by ϕ to an existential variable of V. This set of subgoals and mapping information is call a *MiniCon Description* (MCD), and can be viewed as a generalized bucket. Having considered in advance how each of the variables in the query can interact with the available views, the second phase of the MiniCon Algorithm needs to consider significantly fewer combinations of these generalized buckets.

Using the query and views of the previous example, the MCDs formed during the first phase of the MiniCon Algorithm are:

$V(Y)$	h	φ	G
$V_2(c, d)$	$c\rightarrow c, d\rightarrow d$	$x\rightarrow c, y\rightarrow d$	3
$V_3(f, f)$	$f\rightarrow f, h\rightarrow f$	$x\rightarrow f, y\rightarrow f$	1, 2, 3

where h is a head homomorphism on V_i, $V(\bar{Y})$ is the result of applying h to V_i to create a view specialization, φ is a partial mapping from $Vars(Q_1)$ to $h(Vars(V_i))$ and G is a subset of the subgoals in Q_1 that are covered by some subgoal in $h(V_i)$ under the mapping φ.

In the second phase, the MCDs are combined to produce the query rewritings. In this phase the algorithm considers combinations of MCDs, and for each valid combination it creates a conjunctive rewriting of the query. There is no need for containment testing, as opposed to the Bucket algorithm. The combinations of MCDs considered by the MiniCon Algorithm are those that cover all the subgoals of the query with pairwise disjoint subsets of subgoals. The final maximally contained rewriting is a union of conjunctive queries.

As of 2008, MiniCon has been shown to be the most efficient current algorithm for answering conjunctive queries using conjunctive views.

Chase and Backchase In the presence of a set C of constraints on the relations and the views, Chase and Backchase is a powerful technique for finding *equivalent* rewritings using the views.

The chase [ref Encyclopedia entry Chase] is a well-known technique that can be used to decide containment of queries under constraints. If the chase of Q_1 with C terminates producing a query Q_c then $Q_1 \sqsubseteq_C Q_2$ if and only if $Q_C \sqsubseteq Q_2$. $Q_1 \sqsubseteq_C Q_2$ means that $Q_1 \sqsubseteq Q_2$ on every database instance that satisfies the constraint set C.

For the equivalent rewriting problem under constraints C, Q_1 is given and must effectively find Q_2 such that $Q_1 \equiv_C Q_2$. The algorithm proceeds in two steps. In the *chase* step, it chases Q_1 with C until no more chase steps are possible. This results in a query U called the *universal plan*. The universal plan is a query over the database relations and the views that conceptually incorporate all possible alternative ways to answer Q_1 in the presence of the constraints C. In particular, any minimal conjunctive query equivalent to Q_1 under C is isomorphic to a subquery of U. Hence, to find all minimal rewritings of Q_1, the backchase step of the algorithm searches the finite space of subqueries of U. Specifically, for each subquery Q_s it checks for

equivalence with Q_1 by chasing Q_s with C. Q_s is equivalent to Q_1 if and only if there is a containment mapping from Q_1 into an intermediate chase result of Q_s.

Chase and Backchase applies if the constraint set C includes tuple-generating and equality-generating dependencies. It is sound and complete if C is *weakly acyclic*.

Key Applications

The conceptual framework and techniques developed for query rewriting using views have had significant impact in a number of areas of information systems, especially (but not exclusively) in the areas of information integration and data integration [ref Encyclopedia article Information Integration, ref Encyclopedia article Data Integration]. In particular, as of 2007, data integration theory and systems follow two main architectural approaches for defining and processing queries in a virtual (In a virtual integration system, as opposed to a data warehouse [ref entry Data Warehouse], data resides only in their original locations/sources. The integration system accesses them every time it needs to answer a query) integration system. In *global-as-view* the integrated *global view* (or views) of the disparate data sources is defined (using a standard query language or an ad hoc specification system) in terms of the *local* data sources, and queries are directly expressed in terms only of the global view(s). In this case, processing a query over this global view involves *view expansion*. In *local-as-view* data integration on the other hand, local data sources are expressed in terms of an a priori given *global schema*, i.e., they are expressed as views over the relations of the global schema. Queries are also expressed in terms of the global schema. Since in an integration system the only available data reside in the local sources, answering a query requires rewriting it in terms of the available sources, i.e., rewriting it using the views.

In the above context, sources describe their contents in terms of the local schema. In addition, for sources with limited processing power, their capabilities need to also be taken into account. Since integrated queries are answered by retrieving data from local sources, the data retrieval requests (i.e., the queries) submitted to the sources need to conform to their capabilities. Query rewriting using views has provided the conceptual framework and tools to address the *capability-based rewriting* problem.

Another area of application of the ideas of answering queries using views is query optimization. When the set of relations mentioned in queries overlaps with those mentioned in one or more views, and the views select one or more attributes selected by the query, the view may be *usable* by the query. Moreover, certain indices can also be modeled as materialized views. Using the materialized views may lower the cost of query processing, depending on the available access methods and the selectivity of the involved predicates. Answering queries using views is used to decide view usability and generate the appropriate query plans.

Finally, answering queries using views, especially in the presence of constraints, has influenced the theory of *data exchange*. Data exchange, or data translation, involves taking data described in one, *source*, schema and translating, without loss of information, into data structured under a different, *target*, schema.

Future Directions

Even though significant progress has been made in this area, the applicability and/or efficiency of the developed techniques is often limited. Future work is needed on rewriting techniques that can perform correctly and efficiently for "real" SQL views on real SQL queries, including queries with aggregates, nesting, bag semantics, and user-defined functions. Moreover, the conceptual framework of rewriting using views applies to problems in graph querying, Web service composition, and rewriting of XQuery.

Cross-references

- ▶ Data Integration
- ▶ Global as Views
- ▶ Information Integration
- ▶ Local as views
- ▶ Query Containment
- ▶ View Definition
- ▶ Views

Recommended Reading

1. Abiteboul S. and Duschka O. 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. Cohen S., Nutt W., and Sagiv Y. Containment of aggregate queries. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 111–125.
3. Deutsch A., Popa L., and Tannen V. Query reformulation with constraints. ACM SIGMOD Rec., 35(4), 2006.

4. Duschka O. and Geneserth M. Answering recursive queries using views. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997, pp. 109–116.
5. Halevy A. Answering queries using views: A survey. VLDB J., 10(4):270–294, 2001.
6. Lenzerini M. Data Integration: A theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
7. 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.
8. Lin V., Vassalos V., and Malakasiotis P. MiniCount: Efficient rewriting of COUNT-Queries using views. In Proc. 22nd Int. Conf. on Data Engineering, 2006, pp. 1.
9. Papakonstantinou Y. and Vassalos V. Query Rewriting using semistructured views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 455–466.
10. Pottinger R. and Halevy A. Minicon: A scalable algorithm for answering queries using views. VLDB J., 10(2–3):182–198, 2001.
11. Xu A. and Meral Ozsoyoglu Z. Rewriting XPath queries using materialized views. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 121–132.

Anti-monotone Constraints

Carson Kai-Sang Leung
University of Manitoba, Winnipeg, MB, Canada

Definition

A constraint C is *anti-monotone* if and only if for all itemsets S and S':

$$\text{if } S \supseteq S' \text{ and } S \text{ satisfies } C, \text{ then } S' \text{ satisfies } C.$$

Key Points

Anti-monotone constraints [1,2] possess the following nice property. If an itemset S satisfies an anti-monotone constraint C, then all of its subsets also satisfy C (i.e., C is downward closed). Equivalently, any superset of an itemset violating an anti-monotone constraint C also violates C. By exploiting this property, anti-monotone constraints can be used for pruning in frequent itemset mining with constraints. As frequent itemset mining with constraints aims to find itemsets that are frequent and satisfy the constraints, if an itemset violates an anti-monotone constraint C, all its supersets (which would also violate C) can be pruned away and their frequencies do not need to be counted. Examples of anti-monotone constraints include $min(S.Price) \geq \$20$ (which expresses that the minimum price of all items in an itemset S is at least \$20) and the usual frequency constraint $support(S) \geq minsup$ (i.e., $frequency(S) \geq minsup$). For the former, if the minimum price of all items in S is less than \$20, adding more items to S would not increase its minimum price (i.e., supersets of S would not satisfy such an anti-monotone constraint). For the latter, it is widely used in frequent itemset mining, with or without constraints. It states that (i) all subsets of a frequent itemset are frequent and (ii) any superset of an infrequent itemset is also infrequent. This is also known as the *Apriori property*.

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.

AP@n

▶ Average Precision at n

Applicability Period

Christian S. Jensen[1], Richard T. Snodgrass[2]
[1]Aalborg University, Aalborg, Denmark
[2]University of Arizona, Tucson, AZ, USA

Definition

The *applicability period* (or *period of applicability*) for a modification (generally an insertion, deletion, or update) is the time period that modification is to be applied. Generally the modification is a sequenced modification and the period applies to valid time. This period should be distinguished from *lifespan*.

Key Points

The applicability period is specified within a modification statement. In constrast, the lifespan is an aspect of a stored fact.

This illustration uses the TSQL2 language, which has an explicit VALID clause to specify the applicability period within an INSERT, DELETE, or UPDATE statement.

For insertions, the applicability period is the valid time of the fact being inserted. The following states that Ben is in the book department for 1 month in 2007.

```
INSERT INTO EMPLOYEE
VALUES ('Ben', 'Book')
VALID PERIOD '[15 Feb 2007, 15 Mar 2007]'
```

For a deletion, the applicability period states for what period of time the deletion is to be applied. The following modification states that Ben in fact was not in the book department during March.

```
DELETE FROM EMPLOYEE
WHERE Name = 'Ben'
VALID PERIOD '[1 Mar 2007, 31 Mar 2007]'
```

After this modification, the lifespan would be February 15 through February 28.

Similarly, the applicability period for an UPDATE statement would affect the stored state just for the applicability period.

A *current modification* has a default applicability period that either extends from the time the statement is executed to forever, or when now-relative time is supported from the time of execution to the ever-increasing current time.

Cross-references

- ▶ Current Semantics
- ▶ Lifespan
- ▶ Now in Temporal Databases
- ▶ Sequenced Semantics
- ▶ Temporal Database
- ▶ Time Period
- ▶ TSQL2
- ▶ Valid Time

Recommended Reading

1. Snodgrass R.T. (ed) The TSQL2 Temporal Query Language. Kluwer Academic, 1995.
2. Snodgrass R.T., Developing Time-Oriented Database Applications in SQL, Morgan Kaufmann, 1999.

Application Benchmark

Denilson Barbosa[1], Ioana Manolescu[2], Jeffrey Xu Yu[3]
[1]University of Alberta, Edmonton, AL, Canada
[2]INRIA Saclay–Île de France, Orsay, France
[3]The Chinese University of Hong Kong, Hong Kong, China

Synonyms

Benchmark; Performance benchmark

Definition

An application benchmark is a suite of tasks that are representative of typical workloads in an application domain.

Key Points

Unlike a MICROBENCHMARK, an application benchmark specifies broader tasks that are aimed at exercising most components of a system or tool. Each individual task in the benchmark is assigned a relative *weight*, usually reflecting its frequency or importance in the application being modeled. A meaningful interpretation of the benchmark results has to take these weights into account.

The Transaction Processing Performance Council (TPC) is a body with a long history of defining and published benchmarks for database systems. For instance, it has defined benchmarks for Online Transaction Processing applications (TPC-C and TPC-E), Decision Support applications (TPC-H), and for an Application Server setting (TPC-App).

Other examples of application benchmarks are: the OO1 and OO7 benchmarks, developed for object-oriented databases, and XMark, XBench and TPoX, developed for XML applications.

Cross-references

- ▶ Micro-Benchmarks
- ▶ XML Benchmarks

Recommended Reading

1. Carey M.J., DeWitt D.J., and Naughton J.F. The OO7 benchmark. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 12–21.
2. Gray J. (ed.). The Benchmark Handbook for Database and Transaction Systems, (2nd edn.). Morgan Kaufmann, 1993.

3. Nicola M., Kogan I., and Schiefer B. An XML transaction processing benchmark. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 937–948.
4. Schmidt A., Waas F., Kersten M.L., Carey M.J., Manolescu I., and Busse R. XMark: a benchmark for XML data management. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 974–985.
5. Transaction Processing Performance Council. Available at: http://www.tpc.org/default.asp
6. Yao B.B., Özsu M.T., and Khandelwal N. XBench benchmark and performance testing of XML DBMSs. In Proc. 20th Int. Conf. on Data Engineering, 2004, pp. 621–633.

Application Recovery

David Lomet
Microsoft Research, Redmond, WA, USA

Synonyms

Persistent applications; Fault tolerant applications; Transaction processing; Recovery guarantees; Exactly once execution

Definition

Systems implement application recovery to enable applications to survive system crashes and provide "exactly once execution" in which the result of executing the application is equivalent to a single execution where no system crashes or failures occur.

Historical Background

Application recovery was first commercially provided by IBM's CICS (Customer Information Control System). Generically, these kinds of systems became known as transaction processing monitors (TP monitors) [5,9]. With a TP monitor, applications are decomposed into a series of steps. Each step is executed within a transaction. A step typically consists of reading input state from a database or transactional queue, executing some business logic, perhaps processing user input or reading and writing to a database, and finally, writing state for the next step into database or queue [4]. If a step failure occurs, its transaction is aborted. Since the prior step results are stably stored, the step can be re-executed after system recovery.

Application recovery does not always involve transactions, however, as the early work by Borg et al. demonstrates [6]. More recently, in the web context, TP monitors have been renamed as application servers (app servers). App servers are similar to TP monitors, where state is explicitly managed, frequently by using transactions. Ongoing interest in application recovery is illustrated by the "recovery oriented computing" (ROC) project [3] and the Phoenix project [1].

Foundations

Introduction

Exactly-Once Applications: While application programmers are usually familiar with the problem area for which an application is written, they are frequently also faced with having to deal with "system problems" of reliability and availability. The system goal is to permit applications to achieve "exactly once execution" [2]. For example, an airline reservation system wants to issue exactly the number of tickets a customer requests, instead of no tickets or twice the number.

Types of Failures: Applications cannot survive all forms of failures. If an exactly-once execution produces a "deterministic" failure, then every execution of the application will lead to that same failure. Such deterministic failures are called "hard failures." The failures that software can deal with are called "soft failures." A subsequent execution of the system will usually avoid the state leading to the system failure. Soft failures arise in a number of ways.

1. *Software non-determinism*: A software system is non-deterministic if, when re-executed, it results in a different execution path than a prior execution. Non-determinism can arise when, for example, paths are determined by relative processor speed or the sequence of external events. Such software bugs have been called "Heisenbugs" (hard failures being "Bohrbugs").
2. *Soft hardware failures*: Hardware can also suffer from "Heisenbugs." For example, a transient hardware failure may be triggered by an environmental cause, such as a cosmic ray changing a memory bit, etc.
3. *Operator failures*: Systems occasionally require operator intervention. Operators, being human, make mistakes. An operator is unlikely to make the same mistake at the same point in a subsequent execution.

Recovery is effective because failures are usually "soft" [3,9], which is why database recovery is so successful.

Support for Application Persistence: Applications which guarantee "exactly once" execution are called

"persistent applications" because application state persists across or despite system failures. The traditional method for providing application persistence has been to use a TP monitor. However, new approaches permit implicit state management, in which the application programmer delegates the problem of managing application state to the system infrastructure.

Implicit state management does not require application steps to execute within transactions. It permits applications to be coded more "naturally" (though restrictions exist). Application state is made stable "under the covers." Phoenix [1] does this via logging application non-determinism and replaying the captured non-determinism after a failure to recover application state. CORBA's approach [11], having shorter down time but more costly normal execution, replicates application state so that if part of the system fails, a copy of the state is available elsewhere for continued execution. Implicit state management is discussed here.

Application persistence (recovery) requires different techniques than database recovery: (i) Applications are usually single-threaded and "piece-wise deterministic," while databases execute highly concurrent code. (ii) Application state is frequently distributed, so dealing with distribution is essential. (iii) Databases change state entirely within a transaction, while application state may frequently change outside of a transaction.

Persisting Application State

Implicit state management requires the infrastructure supporting an application to capture application state and make it stable in some way, relieving the programmer of this task. Two approaches have been developed. The approaches are not as different as they appear. Both require that applications be piece-wise deterministic, with clearly identified non-deterministic events that can be "applied" at an application instance to be used in deterministic re-execution that can generate the same state as the original execution.

Recovery: Recovery technology usually assumes that a failing application will be re-activated after a system crash and its activities recovered on the original system. Thus recovery oriented approaches usually capture application state stably (e.g., "on disk") so that it survives a system failure. There are two parts to this:

1. Writing a "snapshot" of the state to stable storage from time to time. It matters when this snapshot, also called a checkpoint, occurs.
2. Stably logging non-deterministic events encountered by the application in arrival order.

Recovery, in which the state is re-created following a system failure, consists of re-installing the latest captured state (checkpoint) followed by re-executing the application by "feeding" events from the log to re-create application state as of the last logged event.

Logging can be either optimistic or pessimistic [7]. Pessimistic logging eagerly makes log records stable, usually as events occur in order to ensure that execution is "exactly once." Optimistic logging defers for a time making events stable, frequently sacrificing exactly once execution. Pessimistic logging can be greatly optimized. Optimistic logging can be constrained so that exactly once execution is assured. This brings these techniques toward some middle ground in performance, though important differences remain.

Replication: Replication technology usually assumes that an application executing on a failing system A will continue execution on a separate system B where a replica of its state has been maintained. The problem for replica oriented approaches is to capture and keep in sync an application's state on these separate systems. There are two generic approaches to this:

1. Designate one system as the primary system, the others as secondaries. Non-deterministic events are sent to the primary system, which then relays them to the secondary in the order that it received them. Replicas thus execute in response to the same sequence of events. A secondary becomes the primary on failure of the primary. If a secondary fails, the primary needs to know this and re-create another secondary.
2. Use atomic broadcast to send non-deterministic events to all systems maintaining replicas "simultaneously" [1]. Atomic broadcast guarantees that replicas receive all messages and in the same order. If there is a failure, then the remaining replicas participating in the atomic broadcast need to know the members of the new group so that the atomic broadcast protocol continues to work correctly, possibly including the creation of a new replica.

Intermediate Approaches: The line between replication and recovery approaches is not always crisp: (i) A recovery system can maintain its log by sending the log records to another system, hence using a second

system as stable storage instead of a disk. This second site might then become the site where the application is subsequently executed should the original site fail. (ii) A second system for a replica might not execute the events forwarded to it immediately, but rather simply "log" them. Then, only if a primary replica fails would it perhaps then "catch up" by executing the stored events.

Thus a second system might serve as a cold standby system (retaining a log of events) or as a hot standby, immediately executing the application in response to the events. Warm standbys are also possible in which an application is executed at a secondary site in a lazy fashion, lagging the state of a primary site, but not by too much. Whether this is should be viewed as recovery or replication is not really important.

Distributed Applications

Many applications are distributed, e.g., a web application might consist of a client component providing the user interface, one or more middle tier components executing business logic, and back end components that typically provide transactions and database functionality. The new problem is to coordinate the states of multiple software components executing different parts of the application [6,7].

1. The state of the set of application components needs to be "causal," i.e., every message receive reflected in the state of some component must always be accompanied by a sender in a state where the message has been sent.
2. Messages (non-deterministic events in a distributed system) must result in an "exactly once" execution by the receiving component.
3. The application must be able to interact with users or other elements outside of the persistence infrastructure, which may not obey the required protocols.
4. The application must be able to interact with transactional elements like databases.
5. Different strategies have different costs and impact the balance between normal run time costs and recovery costs and time-to-recovery.

Contracts for Persistent Components: Providing persistence for components of a distributed application requires an agreed upon set of protocols or "contracts" [1,2] involving component state and message stability, repeated sending of messages, eliminating duplicate messages, etc. The basic contract between persistent components, called the "committed interaction contract" or CIC, places burdens on both sender and receiver of a message at the time it is sent.

The *sender of a message* ensures causality. The sender ensures that it's state as of the time of message send and the message will be persistent earlier than the receiver's state is persisted. Further, the sender must continue to send the message until the receiver acknowledges the message, in order to deal with unreliable networks and crashed receivers, etc. The CIC does not specify how to do this, but the recovery approach usually writes non-deterministic events to a log, and flushes the log when a message is sent should the state include previously received messages since the last log flush.

The *receiver of the message* ensures exactly once execution. The receiver eliminates duplicate messages the sender may send in its effort to provide reliable delivery. The receiver executes in response to a message only if receiver state does not already reflect having received the message and bypasses execution otherwise, returning the same result as produced by the original execution. Finally, the receiver ensures that the state resulting from the message receipt is stable at an appropriate moment. The CIC contract does not specify how to do this, but the receiver might use a table of messages received, or some high water mark for messages when the sender is known, to which it compares each incoming message.

External and Transactional Components: Many web applications involve users entering information at a keyboard, and a database that stores the results of business dealings, e.g., the purchase of a plane ticket. Hence, persistent components must interact with elements outside of the boundaries of the application and its supporting system. This requires new forms of "contracts" in which the main burden is placed on the persistent components to enable an ensemble of elements to achieve exactly once execution, e.g., one purchase of the plane ticket.

External components, including users, may not obey CIC requirements. Hence, external interactions must be limited to ensure exactly once execution. For users, both reads and writes might be exposed as having multiple occurrences. A failure in the middle of a user interaction may require entering data more than once or repeating an output if the system fails between the event and its stable logging. Systems typically

minimize the problem window by immediate logging, with a log flush, in response to external events.

A transactional component only occurs at the edge of an application system, responding to requests for and updates to its data. Transactions can abort, with the transactional component's state reset to remove all effects of the aborted transaction. The persistent component must be prepared to handle transaction aborts at any time. Transactions actually reduce the burden for a persistent component as it need not ensure that its state is stable at each interaction within a transaction. A system crash will eventually lead to the abort of any interrupted transaction. However, at the point where the persistent component requests a transaction commit, it must obey the usual requirements for a sender in a CIC.

Optimizations: Optimizations can reduce the normal runtime cost of providing persistent applications. These exploit additional information known to the application programmer when the application is either written or being deployed.

1. Some components may be read-only, producing no external side effects. So a read only component need do no logging itself, while a calling persistent component can log lazily because the read can be repeated if needed. A functional call component in which the result depends only on the arguments of the call requires no logging as the call can be replayed idempotently.
2. If called components extend the time they stably remember prior calls and results for idempotence, then the calling component need not log to make its state stable prior to each call. It can depend upon the called components in a sequence of calls to capture the call results to enable its deterministic replay.
3. When a component is a "server" for exactly one client, the client can capture what would be nondeterministic calls and their order for the server component, relieving the server from needing to log calls messages. Combining this with item 2 enables persistent components without logging [8,10]. A logless component's state is regenerated by its client replaying the sequence of calls, and it re-executing its own series of calls to other components that have captured the call results. Logless components are easily deployed anywhere, can be freely replicated, and hence make persistence simple, flexible, and low cost.

Checkpoints: Recovery time is shortened via checkpoints. Component state consists of its variables and its execution time stack. By checkpointing component state when there is no execution within the component, the checkpoint can be accomplished solely by capturing its variable values, e.g., checkpoint might be taken transparently during the midst of a return from a call to a component. The cost of the checkpoint and the need for fast recovery time dictate checkpoint frequency.

Discussion

Making declarative the requirements for application persistence, as represented by interaction contracts, makes it easier to provide and optimize exactly once execution by expanding the range of implementation options, including interesting optimizations.

The level of the contract is also important. A CIC could have been derived from reliable messaging instead of more explicitly as repeated sending of messages with duplicate elimination. However, reliable messaging does not describe what happens to messages once they are delivered. By describing the requirements for state stability, etc., it becomes clear that delivery is not, by itself, sufficient. Further, were "persistent reliable messaging" used, this would requiring extra log forces that, can frequently be avoided with the "end-to-end" application persistence protocol.

System provided transparent application persistence is an example of delegating a serious problem to the "system," similar to how transactions delegate concurrency control and recovery to database systems. There is no need for special application programmer consideration, simplifying the application logic, and improving programmer productivity.

Key Applications

Application recovery (persistence) is used wherever exactly once semantics is required. Traditionally, this has been within transaction processing systems, but now more commonly, this involves web applications for e-business, whether for end user customers or business to business dealings. If there are financial or legal requirements for applications, they will be built using some form of application recovery to ensure exactly once execution.

Cross-references

► Transaction

Recommended Reading

1. Barga R., Chen S., and Lomet D. Improving Logging and Recovery Performance in Phoenix/App. In Proc. 20th Int. Conf. on Data Engineering, 2004.
2. Barga R., Lomet D., Shegalov G., and Weikum G. Recovery Guarantees for Internet Applications. ACM Trans. internet Tech., 4(3):289–328, 2004.
3. Berkeley/Stanford Recovery-Oriented Computing (ROC) Project. http://roc.cs.berkeley.edu. October 10, 2008.
4. Bernstein P., Hsu M., and Mann B. Implementing Recoverable Requests Using Queues. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp.112–122.
5. Bernstein P. and Newcomer E. Principles of Transaction Processing. Morgan Kaufmann, 1997.
6. Borg A., Baumbach J., and Glazer S. A message system supporting fault tolerance. In Proc. 9th ACM Symp. on Operating System Principles, 1983, pp. 90–99.
7. Elnozahy E.N., Alvisi L., Wang Y., and Johnson D.B. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. ACM Comp. Surv., 34(3), 2002, pp. 375–408.
8. Frølund S. and Guerraoui R. A Pragmatic Implementation of e-Transactions. In Proc. 19th Symp. on Reliable Distributed Syst., 2000, pp. 186–195.
9. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993.
10. Lomet D. Persistent Middle Tier Components without Logging. In Proc. Int. Conf. on Database Eng. and Applications, 2005, pp. 37–46.
11. Narasimhan P., Moser L., and Melliar-Smith P.M. Lessons Learned in Building a Fault-Tolerant CORBA System. DSN, 2002, pp. 39–44.

Application Server

Heiko Schuldt
University of Basel, Basel, Switzerland

Synonyms

Web application server; Java application server

Definition

An *Application Server* is a dedicated software component in a three-tier or multi-tier architecture which provides application logic (business logic) and which allows for the separation of application logic from user interface functionality (client layer), delivery of data (web server), and data management (database server).

Key Points

Modern information systems, especially information systems on the web, follow an architectural paradigm that is based on a separation of concerns. In contrast to monolithic (single tier) architectures or two-tier client/server architectures where business logic is bundled with other functionality, three-tier or multi-tier architectures consider dedicated application servers which exclusively focus on providing business logic.

In three-tier or multi-tier architectures, application servers typically make use of several middleware services which enable the communication within and between layers. Application servers usually provide the basis for the execution of distributed applications with transactional guarantees on top of persistent data. In large-scale deployments, systems might encompass several instances of application servers (application server clusters). This allows for the distribution of client requests across application server instances for the purpose of load balancing.

Early application servers evolved from distributed TP Monitors. Over time, a large variety of application servers has emerged. The most prominent class consists of Java application servers, either as stand-alone servers or embedded in large software systems.

Cross-references

- ▶ Database Middleware
- ▶ Enterprise Application Integration
- ▶ Java EE
- ▶ Middleware Support for Database Replication and Caching
- ▶ Multi-Tier Architecture
- ▶ Replication in Multi-Tier Architectures
- ▶ Transactional Middleware

Recommended Reading

1. Burke B. and Monson-Haefel R. Enterprise JavaBeans 3.0. O'Reilly, 5th edn., 2006.
2. Jacobs D. Data management in application servers. Datenbank-Spektrum, 8:5–11, 2004.
3. Raghavachari M., Reimer D., and Johnson R.D. The Deployer's problem: configuring application servers for performance and reliability. In Proc. 25th Int. Conf. on Software Eng, May 2003, pp. 484–489.

Application Server Clustering

- ▶ Replication in Multi-Tier Architectures

Application-Centric Interfacing

► Enterprise Application Integration

Application-Level Tuning

Philippe Bonnet[1], Dennis Shasha[2]
[1]University of Copenhagen, Copenhagen, Denmark
[2]New York University, Newyork, NY, USA

Synonyms

Query tuning; Tuning the application interface

Definition

An under-appreciated tuning principle asserts *start-up costs are high; running costs are low*. When applied to the application level, this principle suggests performance of a few bulk operations that manipulate and transport a lot of data rather than many small operations that act on small amounts of data. To make this concrete, this entry discusses several examples and draws lessons from each.

Historical Background

Application-level tuning is about changing the way a task is performed. This entails finding a better algorithm or finding a better way to handle the database. The first is difficult to automate, but the latter goes back to the very first use of the relational databases. Whether on disk or in main memory, databases have generally always performed best when a single statement accesses all and exactly the data needed for a task.

Foundations

Application-level tuning has the nice property that it often is a pure win. Whereas adding or removing indexes often entails a trade-off between insert/delete/update performance and query performance, application-level tuning for the most part improves the performance of certain queries without hurting the performance of others. Rewriting queries to use resources more efficiently also often gives a greater benefit than physical changes.

Assemble Object Collections in Bulk

Object-oriented encapsulation is the principle of shielding the user of an object from the object's implementation. Encapsulation sometimes is interpreted as *the specification is all that counts*. That interpretation can, unfortunately, lead to horrible performance.

The reason is simple. The first design that seems to occur to object-oriented implementers is to make relational records (or sometimes fields) into objects. This has the virtue of generality. Fetching one of these objects then translates to a fetch of a record or a field. So far, so good. But then the temptation is to build bulk fetches from fetches on little objects (the so-called "encapsulation imperative"). The net result is a proliferation of small queries instead of one large query.

Consider for example a system that delivers and stores trade information. Each document type (e.g., a report on a customer account) is produced according to a certain schedule that may differ from one trade type to another. "Focus" information relates trade types to risk analysts. That is, risk analysts may focus on one trade type or another. This gives a pair of tables of the form:

```
Focus(analyst, tradetype)
Tradeinstance(id, tradetype, tradedetail)
```

When an analyst logs in, the system gives information about the trade instances in which he or she would be interested. This can easily be done with the join:

```
select tradeinstance.id, tradeinstance.
tradedetail
from tradeinstance, focus
where tradeinstance.tradetype=focus.
tradetype
and focus.analyst={input analyst name}
```

But if each trade type is an object and each trade instance is another object, then one may be tempted to write the following code:

```
Focus focustypes=new Focus();
Focus.init({input analyst name});
for (Enumeration e=focustypes.elements(); e.hasMore-
Elements();)
{
TradeInstance tradeinst=new TradeInstance();
tradeinst.init(e.nextElement());
tradeinst.print();
}
```

This application program will first issue one query to find all the trade types for the analyst (within the init method of Focus class):

```
select tradeinstance.tradetype
from focus
where focus.analyst={input analyst name}
```

and then for each such type t to issue the query (within the init method of TradeInstance class):

```
select tradeinstance.id, tradeinstance.
tradedetail
from tradeinstance
where tradeinstance.tradetype=t
```

This is much slower than the previous SQL formulation. The join is performed in the application and not in the database server.

The point is not that object-orientation is bad–encapsulation contributes to maintainability. The point is that programmers should keep their minds open to the possibility that accessing a bulk object (e.g., a collection of documents) should be done directly rather than by forming the member objects individually (incurring a start-up cost each time) and then grouping them into a bulk object on the application side.

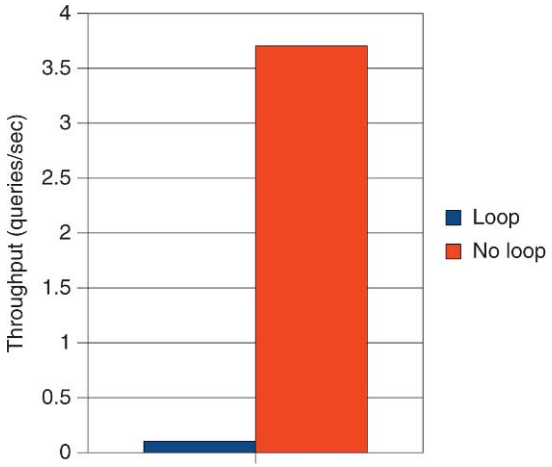
The Art of Insertion

This entry has discussed retrieving data so far. Inserting data rapidly requires understanding the sources of overhead of putting a record into the database:

- As in the retrieval case, the first source of overhead is an excessive number of round trips across the database interface. This occurs if the batch size of inserts is too small. A more radical approach is to assemble all the data to be inserted into a file, load that file into a temporary table, and then insert from that temporary table into the target table. This can improve performance by a factor of 100 or more when tables are large.
- The second issue has to do with the ancillary overhead that an insert causes: updating all the indexes on the table. Even a single index can hurt performance. For this reason, it is often a good idea to add indexes after loading the data.

Key Applications

Application-level tuning has proved useful in every setting where performance is an issue. Whereas index and physical layer tuning can be more easily automated, the benefits of application-level tuning are often greater.



Application-Level Tuning. Figure 1

Experimental Results

Looping hurts

This experiment illustrates the overhead of crossing the application interface. Two hundred tuples are fetched (these are small tuples composed of two integers) using a range query (no loop), or looping on the client-side to fetch one tuple at a time.

Figure 1 traces the throughput observed for this experiment on MySQL 6.0 with a warm buffer. There is an almost two orders of magnitude penalty when looping through the tuples. This is due to the overhead of crossing the application interface 200 times.

Url to Code and Data Sets

Loop experiment: http://www.databasetuning.org/?sec=loop

Cross-references

- ▶ Application server
- ▶ DBMS Interface
- ▶ JDBC
- ▶ ODBC
- ▶ SQL

Recommended Reading

1. Celko J. Joe Celko's SQL for Smarties: Advanced SQL Programming, 3rd edn. Morgan Kaufmann, San Francisco, CA, 2005.
2. Shasha D. and Bonnet P. Database Tuning: Principles, Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Francisco, CA, 2002.
3. Tow D. SQL Tuning. O'Reilly, 2003.

Applications of Emerging Patterns for Microarray Gene Expression Data Analysis

Guozhu Dong[1], Jinyan Li[2]
[1]Wright State University, Dayton, OH, USA
[2]Nanyang Technological University, Singapore, Singapore

Definition

This topic is related to applications of emerging patterns in the bioinformatics field, in particular for in-silico cancer diagnosis by mining emerging patterns from large scale microarray gene expression data.

Key Points

The contemporary gene expression profiling technologies such as cDNA microarray chips and Affymetrics DNA microarry chips can measure the expression levels of thousands even tens of thousands of genes simultaneously. This provides a great opportunity to identify specific genes or gene groups that are responsible for a particular disease, for example, the subtypes of childhood leukemia disease. Reference [3] proposed to use emerging patterns to capture the signature patterns between the gene expression profiles of colon tumor cells and normal cells. This was the first bioinformatics work studying how gene groups and their expression intervals signify the difference between diseased and normal cells. That paper also proposed to design treatment plans to cure the diseased cells by adjusting certain genes' expression level based on the discovered emerging patterns. Reference [2] reported simple rules underlying gene expression profiles of more than six subtypes of acute lymphoblastic leukemia (ALL) patients. The rules are converted from the mined emerging patterns. The rules are also used to construct a classifier (PCL) that reached a benchmark diagnosis accuracy of 96% on an independent test data set. As gene expression data sets contain many attributes, the discovery of emerging patterns by border-differential based algorithms is sometimes slow. To tackle this problem, reference [1] proposed a CART-based approach to discover a proportion of emerging patterns from the high-dimensional gene expression profiling data with a high speed. The ZBDD based approach discussed in the emerging patterns entry of this volume is also fast and can handle large number of genes.

Cross-references

- ▶ Emerging Patterns
- ▶ Emerging Pattern Based Classification

Recommended Reading

1. Boulesteix A.-L., Tutz G., and Strimmer K. A CART-based approach to discover emerging patterns in microarray data. Bioinformatics, 19(18):2465–2472, 2003.
2. Li J., Liu H., Downing J.R., Eng-Juh Yeoh A., and Wong L. Simple rules underlying gene expression profiles of more than six subtypes of acute lymphoblastic leukemia (ALL) patients. Bioinformatics, 19:71–78, 2003.
3. Li J. and Wong L. Identifying good diagnostic genes or genes groups from gene expression data by using the concept of emerging patterns. Bioinformatics, 18:725–734, 2002.

Applications of Sensor Network Data Management

Farnoush Banaei-Kashani, Cyrus Shahabi
University of Southern California, Los Angeles, CA, USA

Synonyms

Applications of Sensor Networks, Applications of Sensor Databases, Applications of Sensor Network Databases

Definition

Sensor networks allow for micro-monitoring of different phenomena of interest in arbitrary physical environments. With this unique capability, sensor networks can capture the events in the real world as they happen in the form of high-resolution spatiotemporal data of various modalities, and provide the opportunity for real-time querying and analysis of the data for immediate response and control. Such functionality is desirable in many classic applications while enabling numerous other novel applications. From the data management perspective, there is a consensus among database researchers that management and analysis of the massive, dynamic, distributed and uncertain data in sensor network applications is going to be one of the new grand challenges for the database community:

- *Sensor information processing will raise many of the most interesting database issues in a new environment, with a new set of constraints and opportunities.*

  – Excerpt from the Lowell Database Research Self-Assessment (By a group of thirty senior

database researchers, *"The Lowell database research self-assessment"*, Communications of the ACM, Volume 48, Issue 5, May 2005.)

Historical Background

Shortly after the introduction of the sensor networks and their potential applications about a decade ago [6], management of the sensor data was recognized as one of the main challenges in realizing the sensor network applications [1].

Foundations

Sensor Databases

One can think of a sensor network as a distributed database that collects, stores and indexes the sensor data to answer the queries received from external users/applications as well as internal system entities. By considering a sensor network as a database, one envisions some of the benefits of the traditional databases potentially for sensor databases; e.g., reduced application development time, convenient multi-user data access and querying with a well-defined generic interface, efficient data reuse, and most importantly data independence. Physical data independence is a particularly beneficial advantage of the sensor database approach, because as compared to the physical layer of the traditional databases the physical infrastructure of the sensor networks is much more sophisticated. With physical data independence in sensor databases, the logical schema of the data exposed to the users is separated from the physical schema that defines the complex and probably changing implementation of the data structures and operations on the physical network. By separating the logical and physical schemas, users/applications are isolated from the typical complications of the distributed data processing in the volatile sensor networks and can focus on designing the logical structure of their queries. Hence, the use of the sensor data is significantly facilitated.

Sensor Database Distinctions

Sensor databases are different from traditional distributed databases in both physical specifications and data characteristics. At the physical level, nodes of the database (i.e., sensor nodes) are severely constrained in resources, such as memory space, storage space, CPU power, and most importantly energy. Moreover, in sensor databases nodes and links of the network are both highly volatile. On the other hand, with sensor devices continuously collecting measurements from the environment, sensor data is naturally very dynamic. Besides, due to inaccuracy of the sensor devices, signal interference, noise, etc., uncertainty is also an inherent characteristic of the sensor data. With such physical and data characteristics, maintaining the illusion of a database is arguably a more difficult objective with sensor databases as compared to that of the traditional distributed databases, and accordingly, requires new data management solutions:

- Database operators should be delay-tolerant, and tolerant to frequent updates of the data
- Query execution should be performed *in-network* for energy efficiency; similarly, data storage and access should be designed for energy efficiency
- Data acquisition plan is required to determine what data to collect
- Sensor query language should be augmented with new operators to specify duration and sampling rate of the data acquisition
- Query execution plan should be dynamically optimized to account for variable access delay and uncertain data availability
- Data uncertainty should be accounted for
- Volatility of the sensor network should be hidden to provide the illusion of a stable database
- Continuous queries should be supported, as sensor networks are primarily used for long-term monitoring
- Meaningful data digests should be maintained to allow for answering historical queries, since data is continuously collected despite the limited space for storage
- Aggregate spatiotemporal queries and range queries should be supported, for energy efficiency [13, 12]
- Approximate queries should be supported, as they are more meaningful with sensor data
- Triggers should be supported for the event-driven monitoring applications

One approach to implement sensor databases is to transfer all data to one or a small number of external base stations, where a traditional database system can be exploited. Alternatively, the data can be stored within the network itself with a balanced and optimal data storage plan. Although with the first approach one can more conveniently employ and extend the data

management solutions applicable with the traditional databases, the second approach, termed *in-network storage*, allows for tighter coupling between query processing, on the one hand, and networking and application semantics, on the other hand. Tight coupling can potentially enable more energy efficient query processing in sensor databases. To evaluate the query processing performance with a particular sensor database implemented with either of these approaches, one can use the standard distributed database performance metrics such as incurred communication cost, query time, indexing time, throughput, load balance among nodes, data update overhead and storage requirements.

Key Applications

As compared to the traditional wireline sensor networks that have been in use for decades, the more recent wireless sensor networks enable low cost and rapid deployment of the sensing network while supporting mobility. With these desirable characteristics due to the wireless technology, recently the standard applications of the sensing networks are revived and new applications that were otherwise unthinkable are identified. The key classes of applications for sensor databases/networks are discussed below.

Environmental Monitoring

Environmental monitoring applications, specifically habitat monitoring [3], are among the earliest applications of the sensor networks. With the habitat monitoring applications, sensors are deployed to monitor animals or plants in their original habitats with most convenience for the scientists and least disturbance for the wildlife. With other environmental applications, sensors can be used to collect earth-science and atmospheric data for environmental explorations, such as the study of the air pollution, global warming, etc., and also early detection and prediction of the natural and man-made disasters, such as hurricanes, wildfires, earthquakes and biological hazards.

Military Intelligence

With rapid deployment, and inexpensive and untethered sensors, wireless sensor networks are well positioned as the tool to collect battlefield data for real-time battlefield intelligence [9]. For instance, wireless sensors can be utilized for geofencing (i.e., deploying a sensor network as a transparent fence to protect an area against unauthorized trespassing), enemy tracking, and battlefield exploration and condition assessment particularly in hazardous environments. In military intelligence applications, the small form-factor, reliability, interoperability and durability of the sensor nodes under severe environmental conditions are particularly critical requirements.

Asset Management

Businesses with large and high-turnover inventories of assets (such as construction companies, utility companies and trucking companies) can benefit from automated asset management systems in improving the utilization of their resources [10]. With automated asset management, sensor networks are deployed to collect real-time data about exact location and condition of an inventory of assets automatically. The collected data provides the opportunity for real-time analysis of the resource usage, which in turn enables timely and optimal decision-making on handling, supply, delivery, storage and other asset management tasks. Various types of sensing devices, such as GPS devices and passive RFID (Radio-Frequency Identification) tags, are applicable with the sensor networks used for asset monitoring.

Building Monitoring

The recent attempt aiming at optimizing the energy performance of the buildings by deep sensing of the building conditions, dubbed BIM (Building Information Modeling) (Federal BIM Program. URL: http://www.gsa.gov/bim/), heavily relies on the sensor network technology. With BIM energy tools enabled by sensing networks, one can monitor, e.g., the temperature and lighting conditions in the building, and accordingly regulate the heating and cooling systems, ventilators and lights dynamically for best energy performance [11]. Also as a safety tool, building sensor networks can detect and report threats, such as existence of the biological agents in the environment as well as physical intrusions.

Automotive

With the new standards such as Dedicated Short-Range Communication (DSRC) designated for vehicle communications, in the near future, cars will be able to communicate information to each other and to the roadside infrastructures. With this capability, while in traffic cars can form a so-called vehicular sensor network, where each car equipped with sensing devices (e.g., camera, thermometer, etc.) acts as a mobile sensor

node. In a vehicular sensor network, cars can share information and analyze the aggregate information about the road conditions, congestions, nearby emergencies, etc., for applications such as collision prevention, congestion avoidance and flow optimization [8].

Healthcare

Sensor networks can effectively improve the accuracy of the patient care and, consequently, the safety of the patients when they become physically incapacitated and require immediate medical attention [7]. Sensor networks allow this by enabling close and automated monitoring of the patient's vital signs. When monitoring is coupled with real-time analysis of the signs, the sensor-enabled healthcare system can alert the right person at the right time to attend to the patient. Such healthcare systems are applicable both at homes of the elderly and at the hospitals.

Industrial Monitoring

Sensors can be used to monitor industrial processes for safety as well as manufacturing optimization [4]. One can also deploy sensors to monitor the condition of the industrial equipments for preventative maintenance and also safety of the operators. Wireline sensors have been in use for a long time in various industry sectors such as oil companies (both upstream and downstream) and chemical plants. Wireless technology and inexpensive sensors has greatly facilitated and extended the use of the sensing networks for process and equipment monitoring, encouraging oil companies, e.g., to develop smart oilfields by equipping the oil wells and other assets with wireless sensors (see e.g., [2]).

Future Directions

With the current trend, sensor networks are being applied with increasingly more complex, large-scale and distributed systems (e.g., the federal intelligent transportation system (Federal ITS Program. URL: http://www.its.dot.gov/)). Such applications demand deployment of large-scale sensor networks and, accordingly, require fully decentralized solutions for sensor data management to achieve scalability.

Data Sets

- CENS data sets. URL: http://research.cens.ucla.edu/portal/page?_pageid = 59,54414&_dad = portal&_schema = PORTAL
- Intel lab data set. URL: http://db.csail.mit.edu/labdata/labdata.html
- Precipitation data set. URL: http://www.jisao.washington.edu/data_sets/widmann/
- IHOP data set. URL: http://www.eol.ucar.edu/rtf/projects/ihop_2002/spol/

Cross-references

- ▶ Sensor Networks
- ▶ Continuous Queries in Sensor Networks
- ▶ Ad-hoc Queries in Sensor Networks
- ▶ In-Network Query Processing
- ▶ Data Acquisition and Dissemination in Sensor Networks
- ▶ Data Aggregation in Sensor Networks
- ▶ Data Storage and Indexing in Sensor Networks
- ▶ Data Uncertainty Management in Sensor Networks
- ▶ Database Languages for Sensor Networks

Recommended Reading

1. Bonnet P., Gehrke J., and Seshadri P. Towards sensor database systems. In Proc. 2nd Int. Conf. on Mobile Data Management, 2001, pp. 3–14.
2. The Center for Interactive Smart Oilfield Technologies. URL: http://cisoft.usc.edu/
3. Cerpa A., Elson J., Estrin D., Girod L., Hamilton M., and Zhao J. Habitat monitoring: Application driver for wireless communications technology. In Proc. SIGCOMM Workshop on Data Communications in Latin America and the Caribbean, 2001.
4. Chong C. and Kumar S.P. Sensor networks: Evolution, opportunities, and challenges. In Proc. IEEE. 91(8), 2003.
5. Culler D., Estrin D., and Srivastava M. Sensor network applications (Cover Feature). IEEE Computer, 37(8), 2004.
6. Estrin D., Govindan R., Heidemann J., and Kumar S. Next century challenges: Scalable coordination in sensor networks. In Proc. 5th Annual Int. Conf. on Mobile Computing and Networking (MOBICOM), 1999, pp. 263–270.
7. Ho L., Moh M., Walker Z., Hamada T., and Su C. A prototype on RFID and sensor networks for elder healthcare. In Proc. 2005 ACM SIGCOMM Workshop on Experimental Approaches to Wireless Network Design and Analysis. Pennsylvania, 2005, pp. 70–75.
8. Lee U., Magistretti E., Zhou B., Gerla M., Bellavista P., and Corradi A. MobEyes: Smart mobs for urban monitoring with a vehicular sensor network. IEEE Wireless Commun, 13(5), 2006.
9. Nemeroff J., Garcia L., Hampel D., and DiPierro S. Application of sensor network communications. In Proc. Military Communications Conference, 2001, pp. 336–341.
10. RFID Journal. URL: http://www.rfidjournal.com/
11. Schmid T, Dubois-Ferrière H., and Vetterli M. SensorScope: Experiences with a wireless building monitoring sensor network.

In Proc. Workshop on Real-World Wireless Sensor Networks, June 2005.
12. Sharifzadeh M. and Shahabi C. Utilizing Voronoi cells of location data streams for accurate computation of aggregate functions in sensor networks. GeoInformatica, 10(1), 2006.
13. Yoon S. and Shahabi C. The Clustered AGgregation (CAG) technique leveraging spatial and temporal correlations in wireless sensor networks. ACM Trans Sensor Netw, 3(1), 2007.
14. Zhao F. and Guibas L. Wireless Sensor Networks: An information processing approach. First Edition, Morgan Kaufmann, San Francisco, CA., July 2004.
15. Center for Embedded Networked Sensing. URL: http://www.cens.ucla.edu/.

Application-to-Application Integration

► Enterprise Application Integration

Approximate Queries in Peer-to-Peer Systems

Wolf Siberski, Wolfgang Nejdl
L3S Research Center, University of Hannover, Hannover, Germany

Synonyms

Top-*k* queries in P2P systems; Aggregate queries in P2P systems

Definition

Peer-to-peer (P2P) networks enable the interconnection of a huge amount of information sources without imposing costs for a central coordination infrastructure. Due to the dynamic and self-organizing nature of such networks, it is not feasible to guarantee completeness and correctness as in traditional distributed databases. Therefore, P2P systems are usually applied in areas where approximate query evaluation, i.e., the computation of a nearly complete and correct answer set, is sufficient. As the most frequent application of querying in P2P is search, many of these algorithms fall into the class of top-*k* query algorithms. Another important case is the approximation of aggregate query results.

Historical Background

P2P networks use approximate querying from the outset. In Gnutella, an unstructured network, the query is distributed in a limited neighborhood only, thus the result is usually not complete. The early top-*k* query algorithms for P2P are based on such unstructured networks. PlanetP [6], a P2P network for information retrieval, employs gossiping to replicate index information among all peers and sends queries to the best peers according to this replicated index. This approach has been extended to a super-peer network in [11]. While the first Distributed Hash Table (DHT) systems such as CHORD and Pastry aim at complete and correct answers, later structured networks approximate the result set, especially P2P networks for information retrieval [2,15]. Frequently, approximate P2P networks build upon algorithms for distributed databases or distributed IR. For example, for source selection PlanetP relies on an extended version of GlOSS [7], Minerva [15] on CORI [4]. Odissea [13] uses either Fagins Algorithm (FA) or a Threshold Algorithm (TA) as top-*k* algorithm, KLEE [9] extends TA, etc. (→ *Top-k Selection Queries on Multimedia Datasets*).

Foundations

The main challenge in P2P approximate query processing is to select the optimal subset of peers to which the query is forwarded. However, the selection criteria for this subset are completely different for top-*k* and aggregate queries. While in top-*k* the goal is to identify the peers holding top objects, approximate aggregate querying algorithms need to find a representative sample of peers. In both cases, each selected peer evaluates the query locally; the respective responses are collected and merged to compute the final result set.

Approximation for Top-*k* Queries

Regardless of the chosen network topology, all distributed top-*k* algorithms consist of the following elements [14]:

Indexing. Determines what is indexed and how index information is collected within the network.

Source selection. Determines the peers a query is sent to.

Result merging. Determines how local result lists are merged to form the final result set.

Unstructured Networks In an unstructured P2P network peers form a random graph. In these

networks, the only available routing strategy is *filtered flooding*, i.e., forwarding the query to selected neighbors. For effective filtering, the peers maintain content indexes. For each neighbor peer, the index allows to look up which kind of content is reachable via this peer. In the case of information retrieval, the index holds term frequencies of these subnets. This index information is built by gossiping: each peer periodically sends the content summaries it holds to its neighbors, where they are merged with the index. Thus, over time each peer gathers more and more accurate information about the whole network. Frequently, bloom filters or hash sketches are used to represent the content summaries. The typical representative for this approach is PlanetP [6]. Evaluations have shown that this algorithm type only scales to several thousands of peers, due to the limitations of filtered flooding. Even with complete index information, the query still usually has to be sent to a high fraction of all peers to reach the peers holding the top-k objects. Also, gossiping induces rather high index maintenance costs.

Hierarchical Networks Hierarchical topologies can overcome some of these limitations. In these topologies, particularly powerful peers form a super-peer backbone. Information sources are not connected with each other, but always assigned to one of the super-peers. This topology is especially suited for adaptation of traditional distributed top-k algorithms: each super-peer acts as coordinating node for its peers. In some systems the peers form a tree-shaped network; this has the advantage that the same aggregation algorithm can be used up to the root peer, but at the price of extremely uneven load distribution. Therefore, the usual approach is to restrict the hierarchy to two levels. In this case, filtered flooding is used to distribute queries within the super-peer backbone. Maintaining the index independently from the actual queries can impose a high overhead; this can be avoided by building a *query-driven index* [2].

Structured Networks Arguably the most efficient peer-to-peer networks are DHTs, where peers form highly structured network topologies. However, they only provide the usual hash table feature, storage and retrieval by key. The DHT maintains lists of top peers for each feature [3,13]. As in the case of hierarchical peers, the index can be improved by considering query statistics [12]. For a query, first these lists are retrieved, and then an established algorithm such as CORI or GlOSS is used for source selection. To retrieve the top-k objects, the TA family of algorithms are the state of the art. TPUT is especially suitable because it limits the retrieval process to three phases: first, the query initiator determines a lower bound for the object score by requesting scores of the top-k objects at each peer. Second, the initiator requests all objects having at least the threshold score. It is guaranteed that all top-k objects are in the returned sets. Finally, the initiator determines these objects and requests the actual content. TPUT has been evolved to an approximate algorithm with probabilistic guarantees in [9].

Approximation for Aggregate Queries

Efficient approximation of aggregate queries in unstructured P2P networks can be done by sampling. Starting at the peer issuing the query, the query travels along a random path to gather the sample. The challenge is to choose this path such that the query is indeed received by a uniform sample of the network. Note that the standard approach (*Markov-Chain random walk*) of selecting each outgoing edge with equal probability does not yield a uniform sample, but favors nodes with high degrees. This can be approximately compensated by scaling down the local peer value with the probability of this peer being selected. To reduce errors due to clustering within the network, the random walk can be modified such that only each ith peer on the path is considered for the sample. [1] shows how usual aggregate queries (COUNT, SUM, AVG) can be computed in this way with low error rates. While gossiping also has be proposed to compute aggregates in unstructured networks, this method does not scale to large networks [10].

An efficient method to compute COUNT queries in DHT networks has been proposed in [10]. This approach relies on locally computed hash sketches which are inserted into the DHT. For hash sketches of length k the actual counting requires $O(k)$ DHT lookups, resulting in $O(k \cdot logn)$ messages.

Key Applications

Top-k queries are used in distributed information retrieval scenarios, such as digital library networks.

An important application for aggregate queries in massively distributed networks is the gathering of network statistics, e.g., to identify security risks or to

monitor performance [8]. Approximate aggregate queries are also gaining importance in the area of sensor network [5], where limitations of the sensor hardware (processor, memory, power supply) are key factors for query algorithm design.

Cross-references

- ► Approximate XML Querying
- ► Peer Data Management System
- ► Top-K Selection Queries on Multimedia Datasets

Recommended Reading

1. Arai B., Das G., Gunopulos D., and Kalogeraki V. Efficient approximate query processing in peer-to-peer networks. IEEE Trans. Knowl. Data Eng., 19(7):919–933, 2007.
2. Balke W.T., Nejdl W., Siberski W., and Thaden U. Progressive distributed top-*k* retrieval in peer-to-peer networks. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 174–185.
3. Bender M., Michel S., Triantafillou P., Weikum G., and Zimmer C. MINERVA: collaborative P2P search. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 1263–1266.
4. Callan J.P., Lu Z., and Croft W.B. Searching distributed collections with inference networks. In Proc. 18th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1995, pp. 21–28.
5. Chu D., Deshpande A., Hellerstein J.M., and Hong W. Approximate data collection in sensor networks using probabilistic models. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 48.
6. Cuenca-Acuna F.M., Peery C., Martin R.P., and Nguyen R.D. Planet P: using gossiping to build content addressable peer-to-peer information sharing communities. In Proc. 12th IEEE Int. Symp. on High Performance Distributed Computing, 2003, pp. 236–246.
7. Gravano L., Garcia-Molina H., and Tomasic A. GlOSS: Text-source discovery over the internet. ACM Trans. Database Syst., 24(2):229–264, 1999.
8. Hellerstein J.M., Condie T., Garofalakis M.N., Loo B.T., Maniatis P., Roscoe T., and Taft N. Public health for the internet (PHI). In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 332–340.
9. Michel S., Triantafillou P., and Weikum G. Klee: A framework for distributed top-k query algorithms. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 637–648.
10. Ntarmos N., Triantafillou P., and Weikum G. Counting at large: Efficient cardinality estimation in internet-scale data networks. In Proc. 22nd Int. Conf. on Data Engineering, 2006, p. 40.
11. Seshadri S. and Cooper B.F. Routing queries through a peer-to-peer infobeacons network using information retrieval techniques. IEEE Trans. Parallel Distrib. Syst., 18(12):1754–1765, 2007.
12. Skobeltsyn G., Luu T., Podnar Z.I., Rajman M., and Aberer K. Web text retrieval with a P2P query-driven index. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007, pp. 679–686.
13. Suel T., Mathur C., Wu J., Zhang J., Delis A., Kharrazi M., Long X., and Shanmugasundaram K. ODISSEA: A peer-to-peer architecture for scalable web search and information retrieval. In Proc. 6th Int. Workshop on the World Wide Web and Databases, 2003, pp. 67–72.
14. Yu C., Philip G., and Meng W. Distributed top-n query processing with possibly uncooperative local systems. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 117–128.
15. Zimmer C., Tryfonopoulos C., and Weikum G. MinervaDL: An architecture for information retrieval and filtering in distributed digital libraries. In Proc. 11th European Conf. on Research and Advanced Technology for Digital Libraries, 2007, pp. 148–160.

Approximate Query Answering

► Approximate Query Processing

Approximate Query Processing

Qing Liu
CSIRO, Hobart, TAS, Australia

Synonyms

Approximate query answering

Definition

Query processing in a database context is the process that deduces information that is available in the database. Due to the huge amount of data available, one of the main issues of query processing is how to process queries efficiently. In many cases, it is impossible or too expensive for users to get exact answers in the short query response time. Approximate query processing (AQP) is an alternative way that returns approximate answer using information which is similar to the one from which the query would be answered. It is designed primarily for aggregate queries such as count, sum and avg, etc. Given a SQL aggregate query Q, the accurate answer is y while the approximate answer is y'. The relative error of query Q can be quantified as:

$$Error(Q) = |\frac{y - y'}{y}|. \quad (1)$$

The goal of approximate query processing is to provide approximate answers with acceptable accuracy in

orders of magnitude less query response time than that for the exact query processing.

Historical Background

The earliest work on approximate answers to decision support queries appears in Morgenstein's dissertation from Berkeley [8]. And the approximate query processing problem has been studied extensively in the last 10 years. The main motivations [3] which drive the techniques being developed are summarized as follows.

First, with the advanced data collection and management technologies, nowadays there are a large number of applications with data sets about gigabytes, terabytes or even petabytes. Such massive data sets necessarily reside on disks or tapes, making even a few accesses of the base data sets comparably slow. In many cases, precision to "last decimal" is not required for a query answer. Quick approximation with some error guarantee (e.g., the resident population in Australia 21,126,700 +/− 200) is adequate to provide insights about the data.

Second, decision support system (DSS) and data mining are popular approaches to analyzing large databases for decision making. The main characteristic of the DSS is that aggregation queries (e.g., count, sum, avg, etc.) are executed on large portion of the databases, which can be very expensive and resource intensive even for a single analysis query. Due to the exploratory nature of decision making, iterative process involves multiple query attempts. Approximate answers with fast response time gives users the ability to focus on their explorations and quickly identify truly interesting data. It provides a great scalability of the decision support applications.

Third, approximate query processing is also used to provide query preview. In most cases, users are only interested in a subset of the entire database. Given a trial query, query preview provides an overview about the data distribution. The users can preview the number of hits and refine the queries accordingly. This prevents users from fruitless queries such as zero-hits or mega-hits. Figure 1 shows an example of query preview interface of NASA EOSDIS (Earth Observing System Data and Information System) project, which is attempting to provide online access to a rapidly growing archive of scientific earth data about the earth's land, water, and air. In the query preview, users select rough ranges for three attributes: area, topic (a menu list of parameters such as atmosphere, land surface, or oceans) and temporal coverage. The number of data sets for each topic, year, and area is shown on preview bars. The result preview bar, at the bottom of the interface, displays the total approximate number of data sets which satisfy the query.

Finally, sometimes network limitation or disk storage failure would cause the exact answers unaffordable or unavailable. An alternative solution is to provide an approximate answer based on the local cached data synopsis.

Foundations

Due to the acceptability of approximate answers coupled with the necessity for quick query response time, approximate query processing has emerged as a cost effective approach for dealing with the large amount of data. This speed-up is achieved by answering queries based on samples or other synopses (summary) of data whose size is orders of magnitude smaller than that of the original data sets.

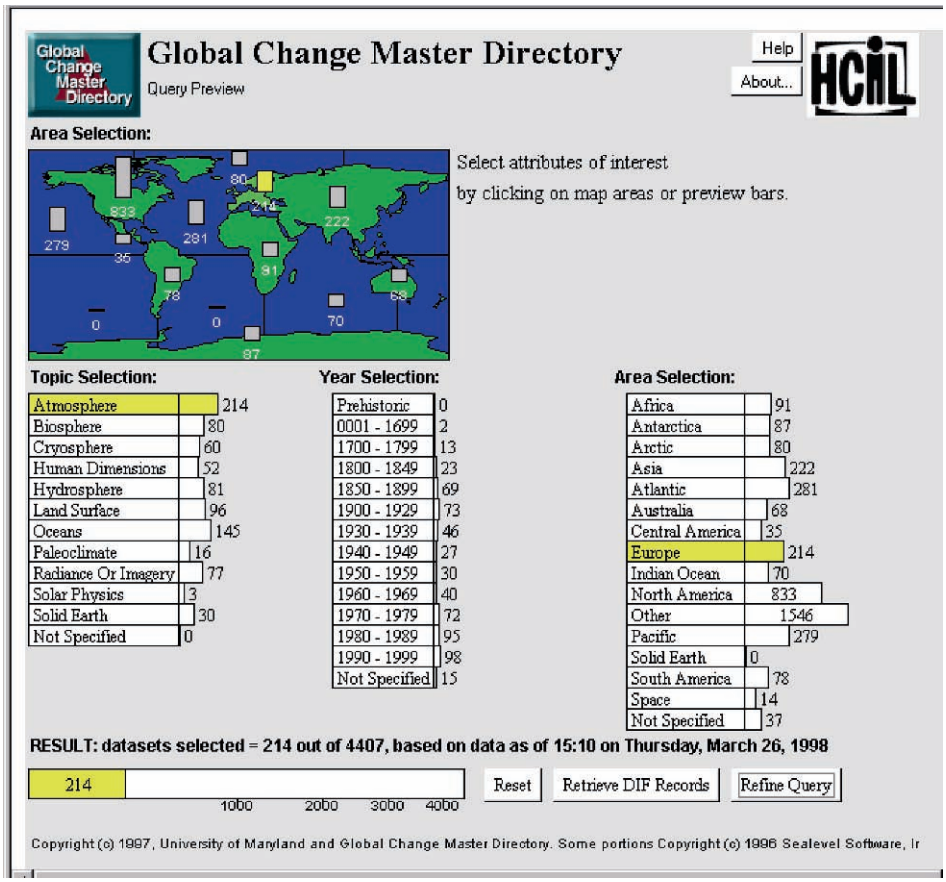
Ioannidis presented the generic flow of approximate query process (Fig. 2) in [5]. Here, data analysis is the overall approach which derives the synopsis from original data to approximate the underlying data distribution. Typically, the *algorithm* partitions the data based on the *distance* function into groups of similar elements, called *buckets*, clusters, patterns, or several other names. The *data elements* that fall in each bucket are then represented by the synopses for *approximation use* which corresponds to the actual purpose of the whole data approximation process. The quality of approximation result can be measured by the *distance* between the synopses and the original data.

There are two basic approaches to achieve approximate query processing: pre-computed synopsis and online query processing.

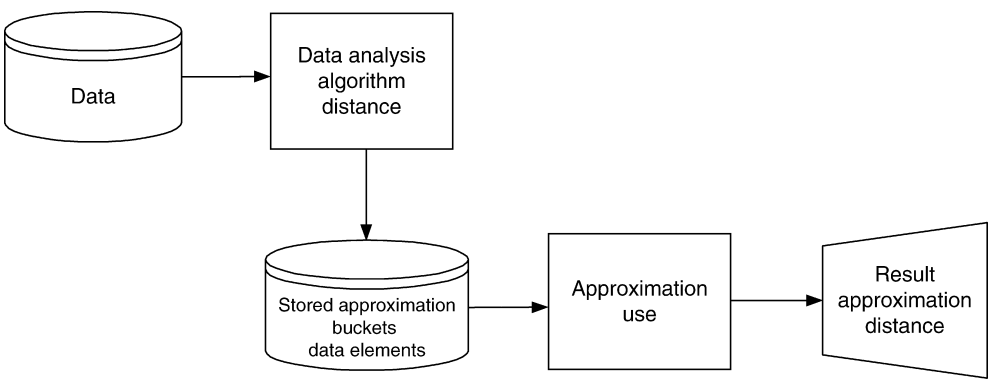
Pre-Computed Synopsis

Approximate query processing using pre-computed synopsis includes two steps: construct synopsis prior to query time, answer query approximately using synopsis at query time.

To provide high accurate query answer, the key issue to construct synopsis is how to represent the underlying data distribution precisely and compactly. Generally, the data distribution can be classified into two groups: uniform distribution and non-uniform distribution. The synopsis for uniform data distribution assumes the objects are distributed uniformly in the data space. For point objects locating in two-dimensional



Approximate Query Processing. Figure 1. NASA EOSDIS interface of query preview.



Approximate Query Processing. Figure 2. Generic flow of approximation process.

space $[U^x_{min}, U^y_{min}] \times [U^x_{max}, U^y_{max}]$, the query result size is estimated as $N \times area(Q)/((U^x_{max} - U^x_{min}) \times (U^y_{max} - U^y_{min}))$, where N is the data set size and $area(Q)$ is the area of window query Q.

There are various techniques developed for non-uniform data distribution. They can also be divided into two groups: parametric and non-parametric. The parametric techniques try to use parameters to catch the original data distributions. Although the models can summarize data distributions with a few descriptive parameters, if the underlying data do not follow any known distributions, or their linear combinations, the model fitting techniques produce inferior results.

The non-parametric techniques use different approaches to summarize the data distributions. Generally, it is possible to classify these techniques into three categories according to the strategies adopted:

1. Sampling techniques
2. Histogram techniques
3. Wavelet techniques

Sampling The basic idea of sampling is that a small random sample of the data often well-represent all the data. Therefore, query would be answered based on the pre-sampled small amount of data and then scaled up based on the sample rate. Figure 3 shows an example where 50% of data are sampled during the pre-computed stage. Given a query "how many Sony laptops are sold in R", the approximate result is "select 2 * sum(*) from S where $S.product = 'SonyLaptop'$", which is 12. In R, the exact answer is 11.

The main issue of sampling method is to decide what sample criteria should be used to select data. The sampling techniques are classified into the following groups [1]:

1. *Uniform sampling*. Data is sampled uniformly
2. *Biased sampling*. A non-uniform random sample is pre-computed such that parts of the database deemed "more important" than the rest
3. *Icicles*. A biased sampling technique that is based on known workload information
4. *Outlier indexing*. Indexing outliers and biased sampling the remaining data
5. *Congressional sampling*. Targeting group by queries with aggregation and trying to maximize the accuracy for all groups (large or small) in each group-by query
6. *Stratified sampling*. Generalization of outlier indexing, Icicles and congressional sampling. It targets minimizing error in estimation of aggregates for the given workload

Sample-based procedures are robust in the presence of correlated and nonuniform data. Most importantly, sampling-based procedures permit both assessment and control of estimation errors. The main disadvantage of this approach is the overhead it adds to query optimization. Furthermore, join operation could lead to significant quality degradations because join operator applied on two uniform random sample can result in a non-uniform sample of the join result which contains very few tuples.

Histograms Histogram techniques are the most commonly used form of statistics in practice (e.g., they are used in DB2, Oracle and Microsoft SQL Server). This is because they incur almost no run-time overhead and

R: Sales

Product	Amount
Sony laptop	1
Sony laptop	1
Sony laptop	2
Sony laptop	3
Sony laptop	4
Dell printer	1
Dell printer	2
LCD monitor	1

S: Sampled sales

Product	Amount
Sony laptop	1
Sony laptop	2
Sony laptop	3
Dell printer	2

Approximate Query Processing. Figure 3. Example of sampling.

produce low-error estimates while occupying reasonably small space.

The basic idea is to partition attribute value domain into a set of buckets and query is answered based on the buckets. The main issues of histogram construction and query are as follows:

1. How to partition data into bucket
2. How to represent data in each bucket
3. How to estimate answer using the histogram

For one-dimensional space, a histogram on an attribute X is constructed by partitioning the data distribution of X into B ($B \geq 1$) buckets and approximating the frequencies and values in each bucket. Figure 4a is an example of original data set and Figure 4b shows its data distribution. Figure 4c is an example of histogram constructed accordingly, where $B = 3$.

If there are several attributes involved in a query, a multi-dimensional histogram is needed to approximate the data distribution and answer such a query. A multi-dimensional histogram on a set of attributes is constructed by partitioning the joint data distribution of the attributes. They have the exact same characteristics as one-dimensional histograms, except that the partition rule needs to be more intricate and cannot always be clearly analyzed because there cannot be ordering in multiple dimensions [9].

To represent data in each bucket, it includes value approximation and frequency approximation. Value approximation captures how attribute values are approximated within a bucket. And frequency approximation captures how frequencies are approximated within a bucket.

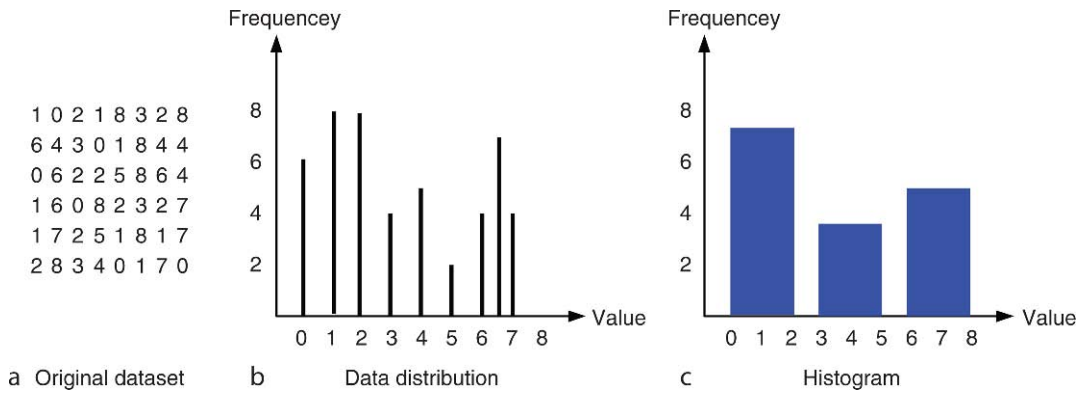
The two main approaches for value approximation are *continuous value assumption* and *uniform spread assumption* [10]. Continuous value assumption only maintains min and max value without indication of how many values there are or where they might be. Under the uniform spread assumption, one also maintain the number of values within each bucket and approximates the actual value set by the set that is formed by (virtually) placing the same number of values at equal distances between the min and max value in multi-dimensional space [6].

With respect to frequency approximation, almost all work deal with *uniform distribution assumption*.

The benefit of a histogram synopsis is that it can be easily used to answer many query types, including the aggregate and non-aggregate queries. However, one of the issues of histogram approach is it is hard to calculate a theoretical error bound. Thus the evaluations on the histogram synopsis usually rely heavily on the experiment results. Further more, histogram-based approaches become problematic when dealing with the high-dimensional data sets that are typical for modern decision support applications. This is because as the dimensionality of the data increases, both the storage overhead (i.e., number of buckets) and the construction cost of histograms that can achieve reasonable error rates increase in an explosive manner.

Wavelet Wavelet is a mathematical tool for hierarchical decomposition of functions using recursive pairwise averaging and differencing at different resolutions. It represents a function in terms of a coarse overall shape, plus details that range from broad to narrow. It is widely used in the signal and image processing.

Matias et al. [7] first proposed the use of Haar-wavelet coefficients as synopsis for estimating the selectivity of window queries. The basic idea is to apply wavelet decomposition to the input data collection to



Approximate Query Processing. Figure 4. Example of histogram.

obtain a compact data synopsis that comprises a select small collection of wavelet coefficients. Figure 5 shows an example of hierarchical decomposition tree of Haar-wavelet. The leaf nodes are original data and non-leaf nodes are wavelet coefficients generated by averaging and differencing from their two children.

Later, the wavelet concept was extended to answer more general approximate queries. The results of recent studies have clearly shown that wavelets can be very effective in handling aggregates over high-dimensional online analytical processing (OLAP) cubes, while avoiding the high construction costs and storage overheads of histogram techniques.

Another important part of the above three technologies for approximate query processing is synopsis maintenance. If the data distribution is not changed significantly, the data synopsis would be updated accordingly to reflect such change. Otherwise, a new data synopsis will be constructed and discard the old one. Refer to the specific techniques for more details.

There are a few other work that do not belong to the above three categories to approximate the underlying data distribution. For example, recently Das et al. [2] present a framework that is based on randomized projections. This is the first work in the context of spatial database which provides probability quality guarantees with respect to query result size approximation.

Online Query Processing

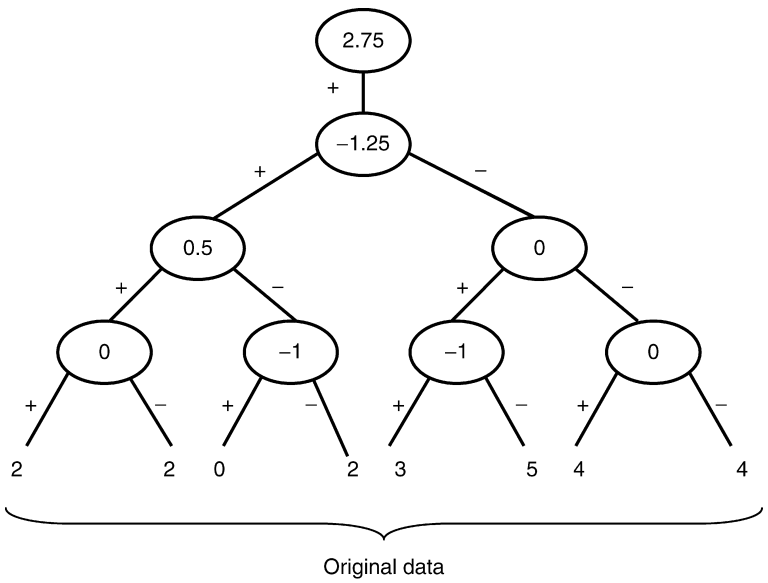
The motivation of online query processing is that the data analysis is to extract unknown information from data. It is an iterative process starting by asking broad questions and continually refining them based on approximate or partial results. Therefore, instead of optimizing for query response time, it needs to balance two conflicting performance goals: minimizing uneventful "dead time" between updates for the users, while simultaneously maximizing the rate at which partial or approximate answers approach a correct answer. Refer to [4] for details.

Compared with pre-computed synopses approach, the advantages of online query processing approach is it does not require pre-processing and progressive refinement of approximate results at runtime can quickly lead to a satisfied results. However, the main obstacles for this technique to be practical is it significantly changes the query processor of current commercial query processing system which is not desirable.

Key Applications

AQP in Relational Data Management

The AQUA system proposed by the Bell lab can support various kinds of approximate queries over the relational database.



Approximate Query Processing. Figure 5. Example of Haar-wavelet hierarchical decomposition.

AQP in Spatial Data Management

Alexandria Digital Library allows user to define spatial queries and returns the approximate number of hits quickly as the initial result and then user can refine the query accordingly.

AQP in Stream Data Management

MIT proposed Aurora as a data stream management system, which virtually supports answering approximate queries over the data stream.

AQP in Sensor Network

The techniques of TinyDB system, proposed by Massachusetts Institute of Technology and UC Berkeley, can lead to orders of magnitude improvements in power consumption and increased accuracy of query results over non-acquisitional systems that do not actively control when and where data is collected.

AQP in Semantic Web Search

For semantic web search, the viewpoints of users performing a web search, ontology designers and annotation designers do not match. This leads to missed answers. Research studies have shown AQP is of prime importance for efficiently searching the Semantic Web.

Cross-references

- ▶ Aggregate Queries in P2P Systems
- ▶ Data Mining
- ▶ Decision Support
- ▶ Histogram
- ▶ On-Line Analytical Processing
- ▶ Query Optimization
- ▶ Sampling
- ▶ Selectivity
- ▶ Wavelets on Streams

Recommended Reading

1. Das G. Sampling methods in approximate query answering systems. In Invited Book Chapter, Encyclopedia of Data Warehousing and Mining, John Wang (ed.). Information Science Publishing, 2005.
2. Das A., Gehrke J., and Riedewald M. Approximation Techniques for Spatial Data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 695–700.
3. Garofalakis M. and Gibbons P. Approximate Query Processing: Taming the TeraBytes: A Tutorial. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001.
4. Hellerstein J., Haas P., and Wang H. Online Aggregation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 171–182.
5. Ioannidis Y. Approximation in Database Systems. In Proc. 9th Int. Conf. on Database Theory, 2003, pp. 16–30.
6. Ioannidis Y. The History of Histograms (abridged). In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 19–30.
7. Matias Y., Vitter J., and Wang M. Wavelet Based Histograms for Selectivity Estimation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 448–459.
8. Morgenstein J. Computer Based Management Information Systems Embodying Answer Accuracy as a User Parameter. PhD Thesis, U.C. Berkeley, 1980.
9. Poosala V. and Ioannidis Y. Selectivity Estimation Without the Attribute Value Independence Assumption. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 466–475.
10. Poosala V., Ioannidis Y., Haas P., and Shekita E. Improved Histograms for Selectivity Estimation of Range Predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 294–305.

Approximate Querying

- ▶ Model-Based Querying in Sensor Networks

Approximate Reasoning

Vilém Novák
University of Ostrava, Ostrava, Czech Republic

Definition

Approximate reasoning is a deduction method which makes it possible to derive a conclusion on the basis of imprecisely characterized situation (quite often using linguistically specified fuzzy IF-THEN rules) and a new information that can also be imprecise. The basic scheme of approximate reasoning is the following:

$$\begin{array}{lll} \text{Condition} & : & \text{IF } X \text{ is } \mathcal{A}_1 \text{ THEN } Y \text{ is } \mathcal{B}_1 \\ & & \cdots\cdots\cdots\cdots\cdots\cdots \\ & & \text{IF } X \text{ is } \mathcal{A}_m \text{ THEN } Y \text{ is } \mathcal{B}_m \\ \text{Premise} & : & X \text{ is } \mathcal{A}' \\ \hline \text{Conclusion} & : & Y \text{ is } \mathcal{B}' \end{array} \quad (1)$$

where "Condition" is a linguistic description consisting of a set of fuzzy/linguistic IF-THEN rules and $\mathcal{A}'$ is a possible modification of antecedent of some of the

former rules. For example, "X is small" can be replaced by "X is very small."

Key Points

The mathematical model of approximate reasoning depends on the way how the linguistic description forming the condition is interpreted (see FUZZY/LINGUISTIC IF-THEN RULES AND LINGUISTIC DESCRIPTIONS).

Let X is $\mathcal{A}'$ be interpreted by a fuzzy set $A' \subseteq U$. If "Condition" is interpreted by a fuzzy relation $R \subseteq U \times V$ then the result of approximate reasoning is a fuzzy set $B' \subset V$ which interprets "Y *is* $\mathcal{B}'$" and which is obtained using the formula

$$B'(y) = \bigvee_{x \in U} (A'(x) \otimes R(x, y)) \qquad (2)$$

where $\otimes$ is a t-norm (product in residuated lattice).

Alternative approximate reasoning method is *perception-based logical deduction*. Its idea consists of finding *perception* of the given measured value of the input $X = x_0$. The perception is an evaluative expression occurring among $\mathcal{A}_1, \ldots, \mathcal{A}_m$ that fits x_0 in the best way. Then the corresponding fuzzy IF-THEN rule is fired and the proper output is derived. More details can be found in [2,3].

Cross-references

- ▶ Fuzzy/Linguistic IF-THEN Rules and Linguistic Descriptions
- ▶ Fuzzy Relation
- ▶ Fuzzy Set
- ▶ Triangular Norms

Recommended Reading

1. Klir G.J. and Yuan B. Fuzzy Sets and Fuzzy Logic: Theory and Applications. Prentice-Hall, New York, 1995.
2. Novák V. and Lehmke S. Logical structure of fuzzy IF-THEN rules. Fuzzy Sets Syst., 157:2003–2029, 2006.
3. Novák V. and Perfilieva I. On the semantics of perception-based fuzzy logic deduction. Int. J. Intell. Syst., 19:1007–1031, 2004.
4. Novák V., Perfilieva I., and Močkoř J . Mathematical Principles of Fuzzy Logic. Kluwer, Boston/Dordrecht, 1999.

Approximate XML Querying

- ▶ Ranked XML Processing

Approximation of Frequent Itemsets

Jinze Liu
University of Kentucky, Lexington, KY, USA

Synonyms

AFI

Definition

Consider an $n \times m$ binary matrix D. Each row of D corresponds to a transaction t and each column of D corresponds to an item i. The (t,i)-element of D, denoted $D(t,i)$, is 1 if transaction t contains item i, and 0 otherwise. Let $T_0 = \{t_1, t_2, \ldots, t_n\}$ and $I_0 = \{i_1, i_2, \ldots, i_m\}$ be the set of transactions and items associated with D, respectively.

Let D be as above, and let $\varepsilon_r, \varepsilon_c \in [0,1]$. An itemset $I \subseteq I_0$ is an approximate frequent itemset AFI(ε_r, ε_c), if there exists a set of transactions $T \subseteq T_0$ with $|T| \geq minsup|T_0|$ such that the following two conditions hold:

1. $\forall i \in T, \frac{1}{|I|} \sum_{j \in I} D(i,j) \geq (1 - \epsilon_r)$;
2. $\forall j \in I, \frac{1}{|T|} \sum_{i \in T} D(i,j) \geq (1 - \epsilon_c)$;

Historical Background

Relational databases are ubiquitous, cataloging everything from market-basket data [1] to genomic data collected in biological experiments [2]. A binary matrix is one common representation of relational databases. Rows in the matrix often correspond to the objects, while columns represent attributes of the objects. The binary value of each matrix entry indicates the presence (1) or absence (0) of an attribute in the object. For example, in a market-basket database, rows represent transactions, columns represent product items, and a binary entry indicates whether an item is contained in the transaction [1]. Frequent itemset mining [1] is a key technique in the analysis of such data. In the binary representation, a *frequent itemset* corresponds to a sub-matrix of 1s, where the itemset (the set of columns) are supported by a sufficiently large number of transactions (set of rows).

While frequent itemsets and the algorithms to generate them have been well studied, the problem is that

the data in real application is often imperfect. In a transaction database, frequent itemsets might be obscured by the vagaries of the market and human behaviors. Items expected to be purchased together by a customer might not appear together in some transactions when one of them is out of stock in the market or overstocked by the customer. In addition, empirical data is subject to measurement noise. For example, Microarray data is often error-prone due to variations in the experimental technology and the stochastic nature of biological processes.

The noise recorded in real applications undermines the ultimate goal of the classical frequent itemset algorithms, i.e., revealing the itemset that is present in a sufficient fraction of transactions (Fig. 1). In fact, when noise is present, the classical frequent itemset algorithms may discover multiple small fragments of the true itemset while missing the true itemset itself. The problem worsens for the most interesting large itemsets since they are more vulnerable to noise.

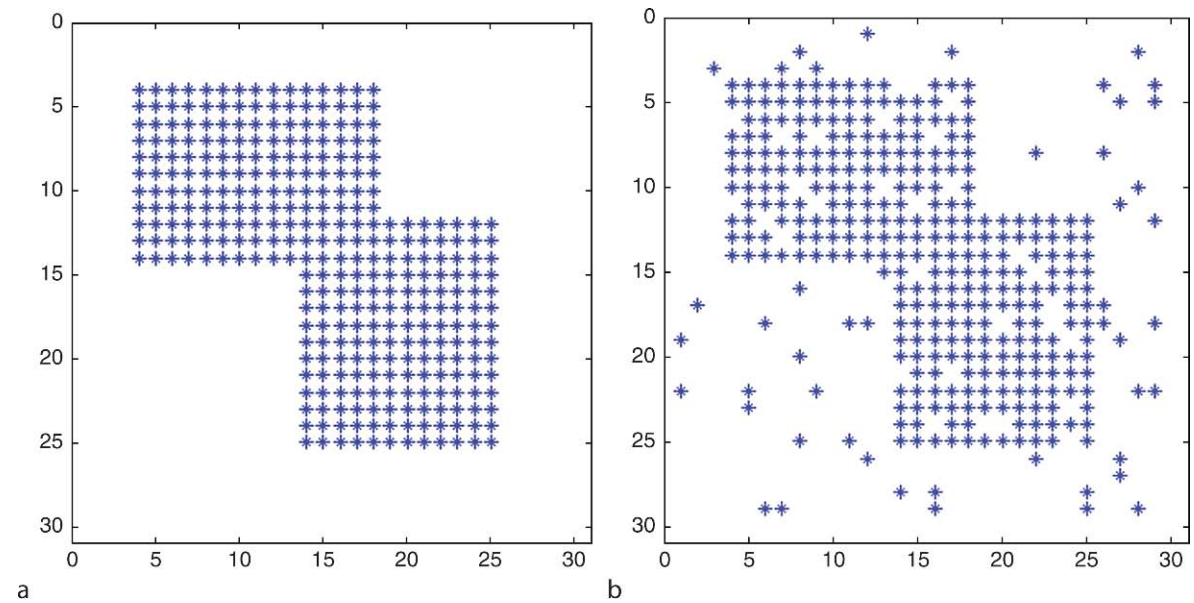
The approximate frequent itemset AFI($\varepsilon_r, \varepsilon_c$) denotes a collection of submatrices of D where the row-wise and column-wise noise levels within each submatrix are below ε_r and ε_c respectively. The classical exact frequent itemset (EFI) is a member of AFI(ε_r, ε_c) where ε_r and ε_c are set to be 0 (Fig. 2). The noise thresholds ε_r and ε_c are usually below 30%. In cases when the noise in either row or column is not restricted, AFI($\varepsilon_r, *$) or AFI($*, \varepsilon_c$) will be used to denote the corresponding families. AFI($\varepsilon_r, *$) corresponds to the same family of itemsets, namely *Error Tolerant Itemset (ETI)*, defined by Yang et al. [8].

Foundations

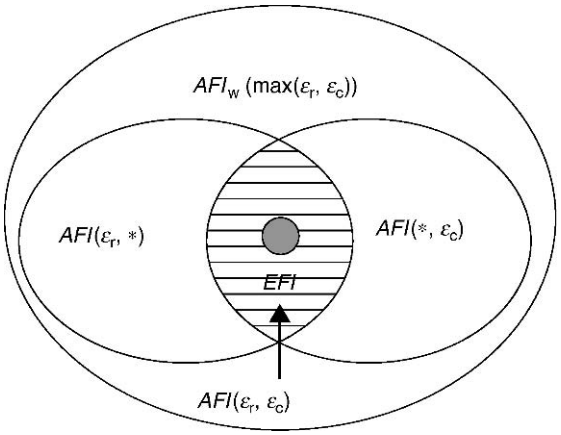
The AFI-mining algorithm [3,4] generalizes the framework of level-wise itemset enumeration. First, the Apriori property of exact frequent itemset mining doesn't hold for AFI when noise is allowed. Instead, conditions under which candidate itemsets can be pruned are established and employed in the AFI algorithm. Secondly, methods that systematically enumerate candidate itemsets without multiple scans of the database are also developed.

Noise-Tolerant Support Pruning

The anti-monotone property of exact frequent itemsets is the key to eliminating the exponential search space in frequent itemset mining [1]. In particular, the anti-monotone property ensures that a $(k+1)$ exact itemset can be pruned if anyone of its k sub-itemsets is not sufficiently supported. However, the allowance of noise may lower the support necessary for the subitemsets of a noise-tolerant itemset. The following theorem suggests a lower bound of support for pruning the candidate itemsets in generating AFIs.



Approximation of Frequent Itemsets. Figure 1. Patterns with and without noise.



Approximation of Frequent Itemsets. Figure 2. Relationships of various AFI criteria.

Theorem 1. *Given a minsup threshold, If a length* $(k+1)$*-itemset* I' *is an AFI*$(\varepsilon_r, \varepsilon_c)$*, for any of its k-sub-itemset* $I \subseteq I'$*, the number of transactions containing no more than* ε_r *fraction of noise must be at least*

$$n \cdot minsup \cdot (1 - \frac{k\epsilon_c}{\lfloor k\epsilon_r \rfloor + 1}) \qquad (1)$$

The noise-tolerant pruning support is defined as the following:

1 Definition

Given ε_c, ε_r and *minsup*, the **noise-tolerant support** for a length-k itemset.

$$minsup_I^k = minsup \cdot (1 - \frac{k\epsilon_c}{\lfloor k\epsilon_r \rfloor + 1})+ \qquad (2)$$

Here $(a)_+ = \max\{a, 0\}$.

The noise-tolerant support threshold is used as the basis of a pruning strategy for AFI mining. The strategy removes supersets of a given I from further consideration when I has support less than $minsup_I^k$. In the special case that $\varepsilon_r = \varepsilon_c = 0$, $minsup_I^k = minsup$, which is consistent with the anti-monotone property of exact frequent itemsets [1].

0/1 Extensions

A transaction t supports a k-itemset I if t contains at least a fraction $1 - \varepsilon_r$ of the items in I. The transaction set of I consists of all the transactions supporting I. Starting with singleton itemsets, the AFI algorithm generates $(k + 1)$-itemsets from k-itemsets in a breadth-first fashion. For each candidate itemset I, transactions t supporting I are generated. The transaction set of a $(k + 1)$ itemset is constructed from the transaction sets of its k-item subsets in one of two different ways, depending on the value of k and ε_r.

0-extension and 1-extension are the two basic steps to be taken for the efficient collection of the supporting transactions. They obtain the supports based on the support of its sub-itemset while avoiding the repeated database scans plaguing the algorithms proposed by [6,8].

Lemma 1 *(1-Extension) If* $\lfloor k \cdot \varepsilon_r \rfloor = \lfloor (k + 1) \cdot \varepsilon_r \rfloor$ *then any transaction that does not support a k-itemset will not support its* $(k + 1)$ *superset.*

The Lemma is based on the fact that if no additional noise is allowed when generating $(k + 1)$ itemset, a transaction does not support a (k-itemset won't support its $(k + 1)$ superset since the number of 1s it contains is always smaller than or equal or $\frac{\lfloor k_*\epsilon \rfloor - 1 + 1}{k+1} < \epsilon$. Thus if $\lfloor k \cdot \varepsilon_r \rfloor = \lfloor (k + 1) \cdot \varepsilon_r \rfloor$ then the transaction set of a $(k + 1)$ itemset I is the intersection of the transaction sets of its length k subsets. This is called a *1-extension.*

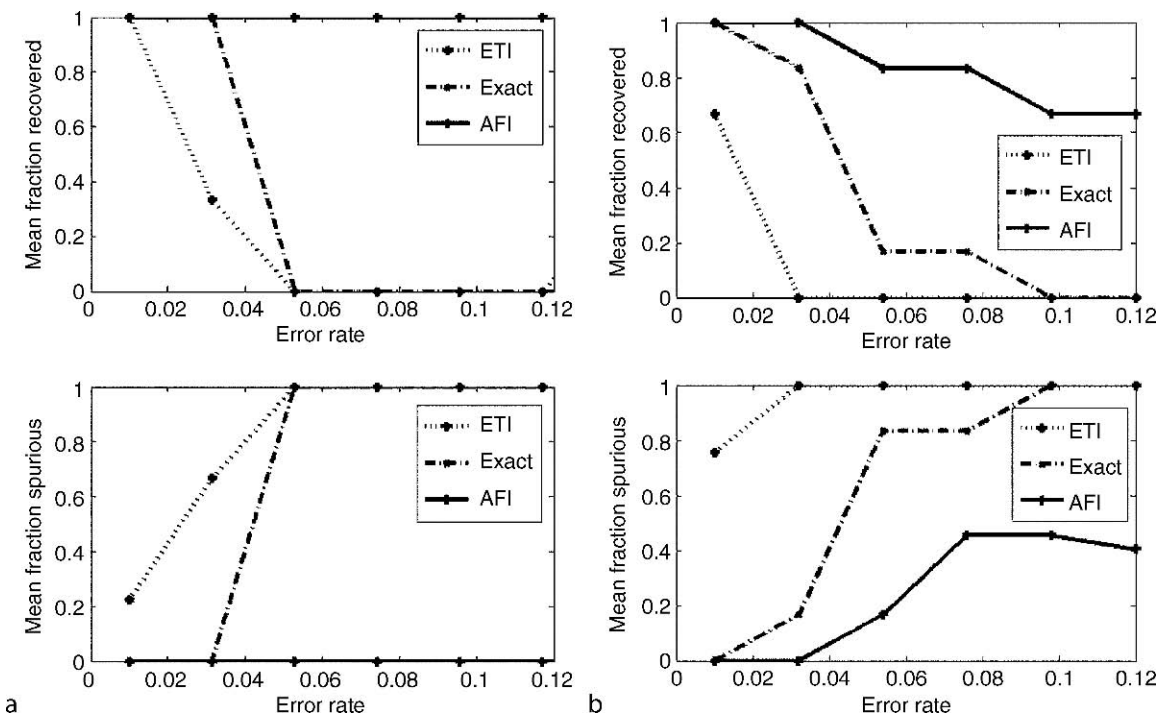
Lemma 2 *(0-Extension) If* $\lfloor k \cdot \varepsilon_r \rfloor + 1 = \lfloor (k+1) \cdot \varepsilon_r \rfloor$ *then any transaction supporting a (k-itemset also supports its* $(k+1)$ *supersets.*

The procedure of 0-extension embodies how noise can be encompassed into a frequent itemset. If additional noise is allowed in a $(k + 1)$-itemset, it is intuitive that a transaction that supports a k-itemset will also support its $k + 1$-item superset, no matter whether the $k + 1$ entry is 1 or 0. To reflect this property, if $\lfloor k \cdot \varepsilon_r \rfloor + 1 = \lfloor (k + 1) \cdot \varepsilon_r \rfloor$, the transaction set of a $(k + 1)$ itemset I is the union of the transaction sets of its length k subsets. This is called a *0-extension.*

Experimental Results

In order to test the quality of the algorithm, data with an embedded pattern and overlaid random errors are generated. The discovered patterns are evaluated against the true patterns which are known in apriori. The methods, exact frequent itemset (EFI), ETI and AFI, are compared in terms of their capabilities in discovering the true patterns in the presence of noise.

Two measures jointly describing the quality are employed. They are "recoverability" and "spuriousness." Recoverability is the fraction of the embedded patterns recovered by an algorithm, while spuriousness is the fraction of the mined results that fail to



Approximation of Frequent Itemsets. Figure 3. Algorithm quality versus noise level.

correspond to any planted cluster. A truly useful data mining algorithm should achieve high recoverability with little spuriousness to dilute the results. A detailed description of the two measures is given in [3].

Multiple data sets were created and analyzed to explore the relationship between increasing noise levels and the quality of the result. Noise was introduced by bit-flipping each entry of the full matrix with a probability equal to p. The probability p was varied over different runs from 0.01 to 0.2. The number of pattern blocks embedded also varied, but the results were consistent across this parameter. The results when one or three blocks were embedded in the data matrix are presented in Fig. 3a,b, respectively.

In both cases, the exact method performed poorly as noise increased. Beyond $p = 0.05$ the original pattern could not be recovered, and all of the discovered patterns were spurious. In contrast, the error-tolerant algorithms, ETI and AFI, were much better at recovering the embedded matrices at the higher error rates. However, the ETI algorithm reported many more spurious results than AFI. Although it may discover the embedded patterns, ETI generates many more patterns that are not of interest, which may overshadow the real patterns of interest. The AFI algorithm consistently demonstrated higher recoverability of the embedded pattern while maintaining a lower level of spuriousness.

Application

AFI mining algorithm can be generally used to find dense 1s blocks in a large binary matrix. The following are example applications.

- E-commerce application: discover approximate frequent itemset in large transactional databases [3,4].
- Biogeographic application: discover regions associated with migration in biogeographic research [3].
- Microarray analysis: find noise tolerant co-expressed patterns.

Cross-references

▶ Association Rule Mining on Streams
▶ Data Mining

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
2. Creighton C. and Hanash S. Mining gene expression databases for association rules. Bioinformatics, 19(1):79–86, 2003.

3. Liu J., Paulsen S., Wang W., Nobel A., and Prins J. Mining approximate frequent itemset from noisy data. In Proc. 2005 IEEE Int. Conf. on Data Mining, 2005, pp. 721–724.
4. Liu J., Paulsen S., Sun X., Wang W., Nobel A., and Prins J. Mining Approximate frequent itemset in the presence of noise: algorithm and analysis. In Proc. SIAM International Conference on Data Mining, 2006, pp. 405–411.
5. Pei J., Tung A.K., and Han J. Fault-tolerant frequent pattern mining: problems and challenges. In Proc. Workshop on Research Issues in Data Mining and Knowledge Discovery, 2001.
6. Seppanen J.K. and Mannila H. Dense itemsets. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery, and Data Mining, 2004, pp. 683–688.
7. UCI machine learning repository. (http://www.ics.uci.edu/mlearn/MLSummary.html).
8. Yang C., Fayyad U., and Bradley P.S. Efficient discovery of error-tolerant frequent itemsets in high dimensions. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001, pp. 194–203.

Apriori Property and Breadth-First Search Algorithms

Bart Goethals
University of Antwerp, Antwerp, Belgium

Synonyms

Monotonicity property; Downward closure property; Levelwise search

Definition

Given a large database of sets of items, called transactions, the goal of frequent itemset mining is to find all subsets of items, called itemsets, occurring frequently in the database, i.e., occurring in a given minimum number of transactions.

The search space of all itemsets is exponential in the number of different items occurring in the database. Hence, the naive approach to generate and count the frequency of all itemsets over the database can not be achieved within reasonable time. Also, the given databases could be massive, containing millions of transactions, making frequency counting a tough problem in itself.

Therefore, numerous solutions have been proposed to perform a more directed search through the search space, almost all relying on the well known Apriori-property. These solutions can be divided into breadth-first search and depth-first search, of which the first is discussed here.

Historical Background

The original motivation for searching frequent itemsets came from the need to analyze so called supermarket transaction data, that is, to examine customer behavior in terms of the purchased products. Frequent sets of products describe how often items are purchased together. In 1993, Agrawal, Imilienski, and Swami introduced this problem, and proposed a first algorithm to solve it. Shortly after that, in 1994, the algorithm was improved and named *Apriori*. The main improvement was to exploit the monotonicity property of the frequency of itemsets, later referred to as the *Apriori property*. The same technique was independently proposed by Mannila, Toivonen, and Verkamo, after which both works were combined in one book chapter [1]. Since then, hundreds of improvements and new algorithms have been developed, many of them relying on the breadth-first search strategy as proposed in the Apriori algorithm.

Foundations

A set of items $\mathcal{I}$ and a database $\mathcal{D}$ of subsets of $\mathcal{I}$, called *transactions*, is given. An *itemset* $I \subseteq \mathcal{I}$ is some set of items; its *support* in $\mathcal{D}$ is defined as the number of transactions in $\mathcal{D}$ that contain all items in I. An itemset is called *frequent* in $\mathcal{D}$ if its support in $\mathcal{D}$ is greater than or equal to a given minimum support threshold σ. The goal is now, given a minimal support threshold σ and a database $\mathcal{D}$, to find all frequent itemsets in $\mathcal{D}$.

Instead of naively generating and counting all possible itemsets, several collections of *candidate itemsets* are generated iteratively, and their supports computed until all frequent itemsets have been generated. Obviously, the size of a collection of candidate itemsets must not exceed the size of available main memory. Moreover, it is important to generate as few candidate itemsets as possible, since computing the supports of a collection of itemsets is a time consuming procedure. In the best case, only the frequent itemsets are generated and counted. Unfortunately, this ideal is impossible generally, which will be shown later in this section.

The main underlying property exploited by most algorithms is that support is monotone decreasing with respect to extension of an itemset.

Property 1. (Apriori Property) *Given a transaction database $\mathcal{D}$ over $\mathcal{I}$, let $X, Y \subseteq \mathcal{I}$ be two itemsets. Then,*

$$X \subseteq Y \Rightarrow support(Y) \leq support(X).$$

Hence, if an itemset is infrequent, all of its supersets must be infrequent. In the literature, this property

is also called the monotonicity property, or also the downward closure property, since the set of frequent itemsets is closed with respect to set inclusion. This property is of crucial importance for all frequent itemset mining algorithms as it allows for pruning large parts of the search space. As soon as an itemset is known to be infrequent, none of its supersets has to be considered anymore.

The Apriori Algorithm

For simplicity, assume that items in transactions and itemsets are kept sorted in their lexicographic order unless stated otherwise.

The itemset mining phase of the Apriori algorithm is given in Figure 1. The notation *X[i]* is used to represent the *i*th item in *X*; the k-*prefix* of a set *X* is the *k*-set {*X*[1],...,*X*[k]}, and $\mathcal{F}_k$ denotes the frequent *k*-sets.

The algorithm performs a breadth-first (levelwise) search through the search space of all sets by iteratively generating and counting a collection of candidate sets. More specifically, a set is candidate if all of its subsets are counted and frequent. In each iteration, the collection C_{k+1} of candidate sets of size $k+1$ is generated, starting with $k=0$. Obviously, the initial set C_1 consists of all items in $\mathcal{I}$ (line 1). At a certain level *k*, all candidate sets of size $k+1$ are generated. This is done in two steps. First, in the *join* step, the union $X \cup Y$ of sets $X, Y \in \mathcal{F}_k$ is generated if they have the same $k-1$-prefix (lines 13–15). In the *prune* step, $X \cup Y$ is inserted into C_{k+1} only if all of its *k*-subsets are frequent and thus, occur in $\mathcal{F}_k$ (lines 16–17).

To count the supports of all candidate k-sets, the database is scanned one transaction at a time, and the supports of all candidate sets that are included in that transaction are incremented (lines 4–7). All sets that turn out to be frequent are inserted into $\mathcal{F}_k$ (line 11).

If the number of candidate sets is too large to remain in main memory, the algorithm can be easily modified as follows. The candidate generation procedure stops and the supports of all generated candidates are counted. In the next iteration, instead of generating candidate sets of size $k+2$, the remaining candidate $k+1$-sets are generated and counted repeatedly until all frequent sets of size $k+1$ are generated and counted.

Although this is a very efficient and robust algorithm, its main drawback lies in its inefficient support counting mechanism. Fortunately, a lot of counting optimizations have been proposed for many different situations.

```
Input: 𝒟, σ
Output: ℱ(𝒟, σ)
 1:  C_1 := {{i} | i ∈ ℐ}
 2:  k := 1
 3:  while C_k ≠ {} do
 4:    for all transactions (tid,I) ∈ 𝒟 ← do
 5:      for all candidate sets X ∈ C_k do
 6:        if X ⊆ I then
 7:          Increment X. support by 1
 8:        end if
 9:      end for
10:   end for
11:   ℱ_k := {X ∈ C_k | X.support ≥ σ}
12:   C_{k+1} := {}
13:   for all X,Y ∈ ℱ_k, such that X[i] = Y[i]
14:      for 1 ≤ i ≤ k − 1, and X[k] < Y[k] do
15:      I := X ∪ {Y[k]}
16:      if ∀ J ⊂ I, |J| = k: J ∈ ℱ_k then
17:         Add I to C_{k+1}
18:      end if
19:    end for
20:    Increment k by 1
21:  end while
```

Apriori Property and Breadth-First Search Algorithms.
Figure 1. Apriori.

Optimizations

A lot of other algorithms proposed after the introduction of Apriori retain the same general structure, adding several techniques to optimize certain steps within the algorithm. Since the performance of the Apriori algorithm is almost completely dictated by its support counting procedure, most research has focused on that aspect of the Apriori algorithm. Here, only four out of more than hundreds of improvement proposals are outlined, but at least, these four represent the most influential and largest jumps forward.

Item Reordering One of the most important optimizations which can be effectively exploited by almost any frequent set mining algorithm, is the reordering of items.

The underlying intuition is to assume statistical independence of all items. Then, items with high frequency tend to occur in more frequent sets, while low frequent items are more likely to occur in only very few sets.

For example, in the case of Apriori, sorting the items in support ascending order improves the distribution of the candidate sets within the used data

structure [4]. Also, the number of candidate sets generated during the join step can be reduced in this way. Unfortunately, until now, no results have been presented on an optimal ordering of all items for any given algorithm and only vague intuitions and heuristics are given, supported by practical experiments.

Partition As the main drawback of Apriori is its slow and iterative support counting mechanism, Savasere et al. [12] proposed the Partition algorithm.

The main novelty in the Partition algorithm, compared to Apriori, is that the database is partitioned into several disjoint parts and the algorithm generates for every part all sets that are relatively frequent within that part. The parts of the database are chosen in such a way that each part fits into main memory, allowing for much more efficient counting mechanisms. Then, the algorithm merges all relatively frequent sets of every part together. This results in a superset of all frequent sets over the complete database, since a set that is frequent in the complete database must be relatively frequent in one of the parts. Finally, the actual supports of all sets are computed during a second scan through the complete database.

Sampling Another technique to solve Apriori's slow counting is to use sampling as proposed by Toivonen [13].

The presented Sampling algorithm picks a random sample from the database that fits in main memory, then finds all relatively frequent patterns in that sample, and finally verifies the results with the rest of the database. In the cases where the sampling method does not produce all frequent sets, the missing sets can be found by generating all remaining potentially frequent sets and verifying their supports during a second pass through the database. The probability of such a failure can be kept small by decreasing the minimal support threshold. However, for a reasonably small probability of failure, the threshold must be drastically decreased, which can cause a combinatorial explosion of the number of candidate patterns. Nevertheless, in practice, finding all frequent patterns within a small sample of the database can be done very fast using fast in-memory support counting techniques. In the next step, all true supports of these patterns must be counted after which the standard levelwise algorithm could finish finding all other frequent patterns by generating and counting all candidate patterns iteratively. It has been shown that this technique usually needs only one more scan resulting in a significant performance improvement [13].

Concise Representations If the number of frequent sets for a given database is large, it could become infeasible to generate them all. Moreover, if the database is dense, or the minimal support threshold is set too low, then there could exist a lot of very large frequent sets, which would make sending them all to the output infeasible to begin with. Indeed, a frequent set of size k includes the existence of at least $2^k - 1$ frequent sets, i.e., all of its subsets. To overcome this problem, several proposals have been made to generate only a concise representation of all frequent sets for a given database such that, if necessary, the frequency of a set, or the support of a set not in that representation can be efficiently determined or estimated [2,5–11].

Key Applications

The Apriori property and the breadth-first search algorithms have broad applications in mining frequent itemsets and association rules. Please refer to the entries of frequent itemset mining and association rules. The Apriori property and the breadth-first search algorithms can be extended to mine sequential patterns. Refer to the entry of sequential patterns. Moreover, they can also be used to tackle other data mining problems such as density-based subspace clustering.

Experimental Results

So far, several hundreds of scientific papers present different techniques and optimizations for frequent set mining and it seems that this trend is keeping its pace. For a fair comparison of some of these algorithms, a contest was organized to find the best implementations in order to understand precisely why and under what conditions one algorithm would outperform another [3]. Although there were no clear winners, one of the rather surprising results of that contest was that the original Apriori algorithm often still performs among the best.

Cross-references

▶ Association Rule Mining on Streams
▶ Closed Itemset Mining and Non-Redundant Association Rule Mining
▶ Frequent Itemsets and Association Rules
▶ Frequent Itemset Mining with Constraints

Recommended Reading

1. Agrawal R., Mannila H., Srikant R., Toivonen H., and Verkamo A. Fast discovery of association rules. In Advances in Knowledge Discovery and Data Mining. U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (eds.). MIT, Cambridge, MA, USA, 1996, pp. 307–328.
2. Bayardo J. and Roberto J. Efficiently mining long patterns from databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp. 85–93.
3. Bodon F. A fast apriori implementation. In Proc. ICDM Workshop on Frequent Itemset Mining Implementations, vol. 90 of CEUR Workshp Proceedings, 2003.
4. Borgelt C. and Kruse R. Induction of association rules: apriori implementation. In Proc. 15th Conference on Computational Statistics, 2002, pp. 395–400.
5. Boulicaut J.F., Bykowski A., and Rigotti C. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. Data Min. Knowl. Discov., 7(1):5–22, 2003.
6. Bykowski A. and Rigitti C. A condensed representation to find frequent patterns. In Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2001, pp. 267–273
7. Calders T. and Goethals B. Mining all non-derivable frequent itemsets. In Principles of Data Mining and Knowledge Discovery, 6th European Conf., 2002, pp. 74–85.
8. Calders T. and Goethals B. Minimal k-free representations of frequent sets. In Principles of Data Mining and Knowledge Discovery, 7th European Conf., 2003, pp. 71–82
9. Gunopulos D., Khardon R., Mannila H., Saluja S., Toivonen H., and Sharma R. Discovering all most specific sentences. ACM Trans. Database Syst., 28(2):140–174, 2003.
10. Mannila H. Inductive databases and condensed representations for data mining. In Proc. 14th Int. Conf. Logic Programming, 1997, pp. 21–30.
11. Pasquier N., Bastide Y., Taouil R., and Lakhal L. Discovering frequent closed itemsets for association rules. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 398–416.
12. Savasere A., Omiecinski E., and Navathe S. An efficient algorithm for mining association rules in large databases. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 432–444.
13. Toivonen H. Sampling large databases for association rules. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 134–145.

ARBAC97

► Administration Model for RBAC

Architecture-aware Database System

► Architecture-Conscious Database System

Architecture-Conscious Database System

John Cieslewicz, Kenneth A. Ross
Columbia University, New York, NY, USA

Synonyms

Architecture-sensitive database system; Architecture-aware database system; Hardware-conscious database system

Definition

Database systems designed with awareness of and a sensitivity to the underlying computer hardware are "architecture-conscious." In an architecture-conscious database system implementation, the performance characteristics of computer hardware guide algorithm and system design.

Historical Background

Database system implementation has been, in varying ways, architecture conscious from the advent of the relational database. For instance, System R [2], an early relational database system prototype included the number of I/Os as a cost metric in its optimizer. At a very high level, the implementers of System R included the characteristics of the underlying hardware in their analysis. This trend has continued with growing attention paid by the database research community to the effects of hardware technology on database performance. Architecture-conscious design took on greater importance as processor speeds improved by four orders of magnitude between 1980 and 2005, while memory latency improved by less than a single order of magnitude. Because of this performance gap, memory accesses, which are central to any database workload, became relatively expensive, requiring database researchers to design database algorithms with this hardware limitation in mind. New computational features have been added to microprocessors, including SIMD instructions, branch prediction, and memory prefetching. These techniques help to improve single threaded performance and instruction level parallelism (ILP), which is the simultaneous processing of instructions from the same thread. In order to take maximal advantage of many of these features, databases must be designed with them in mind.

The introduction of chip multiprocessors (CMP), in particular, has created new challenges and opportunities

for improving database performance. Due to problems with power usage, heat dissipation, and diminishing single threaded performance returns from increasingly complex logic to exploit additional ILP, beginning early in the 2000s chip architects switched their design emphasis from faster clock rates to increased on chip parallelism. It is expected that the degree of parallelism supported by CMPs will continue to increase, making it imperative to design database operators for effective on-chip parallelism. In addition to single threaded performance, achieving good performance on chip multiprocessors requires awareness of thread level parallelism (TLP), the simultaneous processing of instructions from multiple instruction streams. All of these features provide additional opportunities for architecture-conscious database system design.

Foundations

The underlying hardware architecture of which an architecture-conscious database system must be aware includes all computer components. For database systems, the most critical components are persistent storage, the memory hierarchy, and the microprocessor. Research in this field is conducted using architecture simulators and with real hardware. Both are valuable tools, as simulators can test new techniques on hypothetical future hardware designs and using real hardware yields results that are applicable to systems in production now. This article focuses on identifying and explaining the architectural features that have performance implications for databases, and providing high-level design guidelines for database systems implementers using these architectures. A more detailed description of specific database implementation techniques is given in [6].

Persistent Storage

As mentioned above, the impact of storage subsystems has been part of database implementation from the beginning, and many database textbooks describe the costs of various relational operators in terms of the number of I/Os required. Accessing secondary storage incurs significant latency compared with accessing data in memory. For magnetic disk based storage systems, sequential access is favored because each read and write requires a large, fixed cost (rotational and seek latency) before the relatively fast reading or writing. Because of this property, databases are often optimized for sequential reads and writes to storage, e.g., reading entire buffer pages rather than individual records. Magnetic disks, however, are not the only type of persistent storage. Flash memory storage devices also provide high density persistent storage, but have different properties than magnetic disks. As of 2007, flash memory does not have density comparable to magnetic disks, but the technology is improving. Flash memory, unlike a magnetic disk, has no mechanical parts, uses less power, and supports a higher number of random I/Os per second. These different characteristics require new thinking about the way a database system uses persistent storage. For instance, the read, write, and erase properties of flash memory make is less suitable for update-in-place database operations than magnetic disks. Changes to page layout and logging have been introduced to overcome this difference [11]. Flash memory's support for more random I/Os per second, also leads to changes in cost-performance metrics such as the five-min rule [7].

Main Memory and Cache Optimizations

Memory density has increased while prices have fallen, resulting in affordable systems in which a database's entire working set can be held in memory. For such systems, I/O latency no longer dominates. At the same time, processor speeds have increased at a much faster rate than memory latency, making a data load from main memory relatively more expensive. Computer architects have combated this problem by adding caches to processors. A cache is a small, but very fast memory on the same chip as the processor. The cache provides a processor with fast access to frequently used data (temporal locality) or data that resides near recently used data (spatial locality). When data is found in the cache, it is called a *cache hit*, but when data is not found in the cache this is a *cache miss* and main memory must be accessed. Accessing main memory is a longer latency operation that can stall the processor's pipeline. On database workloads, cache misses for data and instructions have been shown to account for a significant portion of execution time [1]. Therefore, data structures and query execution techniques that make more efficient use of limited cache resources are complementary improvements that reduce cache misses and improve performance.

In addition to improving cache use, the challenge posed by the memory bottleneck can be overcome by overlapping memory latency with computation. Prefetching facilitates that overlapping by attempting to load data into the cache before it is needed. Prefetching

can be controlled both in hardware and software. In hardware prefetching, the processor attempts to identify access patters, such as a sequential scan, and load data ahead of when it is needed. In software prefetching, the programmer inserts prefetch instructions into a program to provide the hardware with hints about what data should be loaded into the cache. Software prefetching must be used when there is no pattern to the memory accesses. Chen et al. have improved the performance of tree-based index searches and hash joins by using prefetch instructions [3,4].

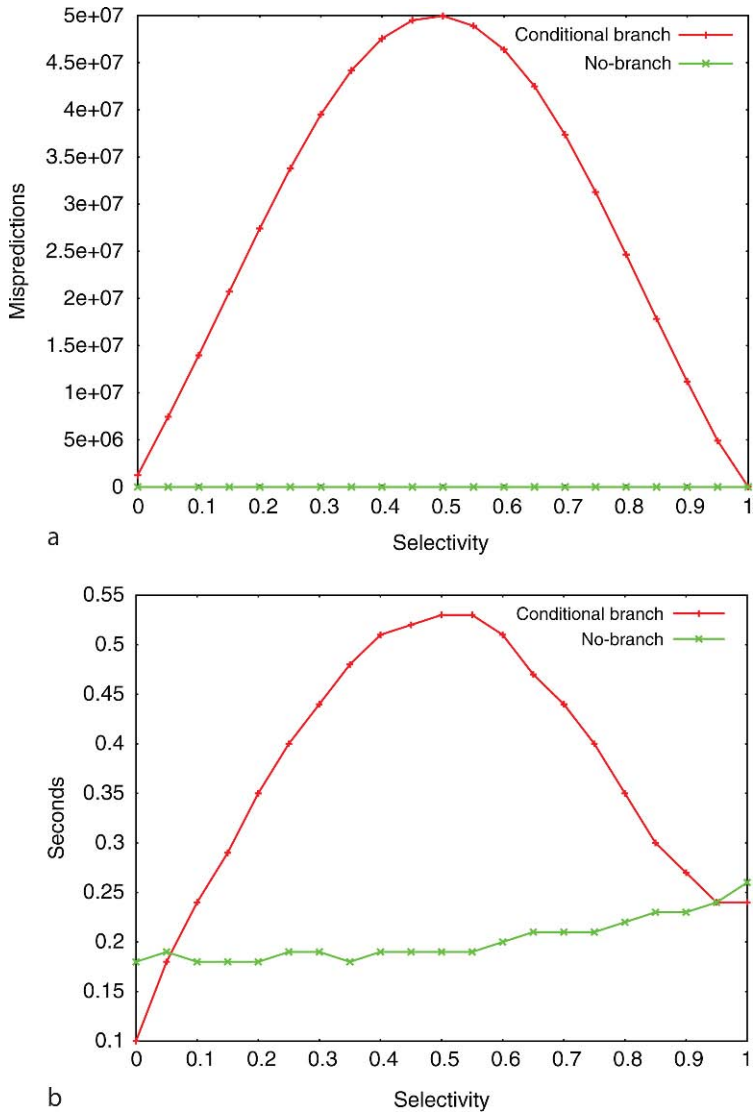
Microarchitecture Optimizations

Conditional branches are a performance hazard on many microarchitectures. A conditional branch represents a control dependency, after which the processor does not immediately know which instruction to execute. On architectures with long pipelines, the outcome of a conditional test is not known for many cycles. In the interim, the processor must either stop issuing instructions or guess the outcome of the condition. This control dependency can limit instruction level parallelism (ILP) because the compiler is limited in its ability to reorder instructions around a branch and the processor is unsure which instructions to issue after a branch instruction. Many modern processors attempt to guess the outcome of the branch using hardware branch prediction, so that instructions continue to be issued in spite of the branch. Branch prediction attempts to identify patterns in the recent history of branch outcomes in order to guess the present branch's outcome correctly. If the branch is predicted correctly, the branch causes no delay in the pipeline. When the prediction is incorrect, the pipeline must be flushed and restarted because invalid instructions have been issued. This cost is the "branch misprediction penalty" and is related to the depth of the pipeline. When there is no pattern to the branching, branch prediction fails up to half the time, and the misprediction penalty degrades performance. In database workloads, where branches are often data dependent, branch misprediction has a noticeable impact on performance [1]. Figure 1a shows the number of mispredictions of performing a simple range selection on an array of 100 million uniformly distributed four byte integers on a 2.4 GHz Intel Core 2 Duo processor. The graphs in Fig. 1 show data for a selection implementation using a conditional branch and a "no-branch" implementation in which the conditional branch's control dependency has been changed to a data dependency [12]. The maximum number of mispredictions occur in the conditional branch implementation when the selectivity is 0.5 because the branch prediction is wrong half of the time. This high rate of branch misprediction impacts performance, as shown in Fig. 1b. In contrast, the implementation with no conditional branch is less sensitive to the selectivity, with its modest rise in execution time (Fig. 1b) attributed to the cost of writing more output as the selectivity increases. Ross extends the query optimizer's cost model to take the cost of branch misprediction into account and demonstrates techniques for evaluating selection conditions using varying numbers of conditional branches [12].

Modern microarchitectures feature special vector operations, also known as *single instruction multiple data* (SIMD) instructions. These instructions operate on wide registers containing multiple variables, enabling data parallelism. For example, a 128 bit SIMD register can contain 16 `char` data types or four 32-bit integers. A wide range of mathematical, logical, and data movement operations can be applied in parallel to those variables using a single instruction. SIMD instructions can be used to improve the data parallelism present in database execution, resulting in higher performance [14]. For some architectures, such as the Cell Broadband Engine, which are highly optimized for SIMD processing, SIMD-izing database operations is crucial to achieving good performance [9]. SIMD instructions can be used to transform control dependencies into data dependencies. For instance, when using scalar instructions, selection is commonly performed with a comparison operation followed by a conditional branch. In contrast, with SIMD instructions, a selection is performed by applying a SIMD comparison operation to multiple elements in parallel. The result of this operation is a bit-mask that can be used to select the elements that pass the condition using other instructions. Transforming a control dependency into a data dependency can also be accomplished with predicated scalar instructions such as a *conditional move* instruction. Regardless of how the control dependency is eliminated, doing so will improve ILP, particularly if the branch is difficult to predict, as is the case in the example shown in Fig. 1.

On-Chip Parallelism

A chip multiprocessor (CMP) provides multiple on chip hardware thread contexts, often in the form of multiple



Architecture-Conscious Database System. Figure 1. Number of branch mispredictions and execution time for a one-sided range selection with varying selectivity. The red, "Conditional Branch," line is an implementation using a conditional branch, while the green, "No-Branch," line is an implementation where the conditional branch's control dependency has been replaced with a data dependency [8].

processor cores on a single die. A processor core's pipeline may be shared by different instruction streams, a design known as simultaneous multithreading (SMT). Some CMPs have both multiple cores and multiple threads per core. As of 2007, the most cores available on a commodity processor was eight, with each core supporting four threads for a total of 32 threads per CMP.

Hardware support for thread level parallelism (TLP) in a CMP differs from a symmetric multiprocessor (SMP) in that a CMP is one processor die with multiple processor cores and an SMP is a system composed of multiple processors sharing one main memory. This difference has two important implications. First, when data is shared among two or more processors in an SMP system, modifications to that data by any processor is coordinated by a cache coherency protocol, which uses the system bus to communicate information about changes to shared data. The system bus is significantly slower than the processors and if a large amount of data is shared among the processors or it is frequently updated, cache coherency overhead can significantly

impact performance. In contrast, because all of the cores of a CMP are on the same die, communication between the cores can occur at chip speeds, greatly reducing the coherency overhead of shared data. Second, in a SMP system, although the main memory is shared, each processors' cache hierarchy is separate. In contrast, many CMPs share some cache levels among the processor cores or threads. Because some cache resources may be shared, designing threads to share data in the cache can be beneficial provided that coordinating the sharing does no introduce large amounts of overhead. These differences have implications for architecture-conscious database systems such as the challenge of designing core database operations that take advantage of the on-chip parallelism afforded by chip multiprocessors in a manner that will scale as CMPs provide more hardware thread contexts.

Chip multiprocessors present opportunities and challenges for improving database performance [8]. Finding parallelism in database systems is not difficult. Most database systems have long supported concurrent queries from different users. Other forms of parallelism can be found within a single query or within a single operator. Parallelism alone does not guarantee optimal performance on a CMP. An architecture-conscious database system on a CMP must exploit the on-chip parallelism in a manner that uses the parallel resources to maximum effect. For instance, Cieslewicz and Ross find that balancing parallelism, cache sharing, and thread communication is key to achieving good performance when computing aggregates using a CMP [5]. Chip multiprocessors can also help alleviate some of the problems associated with the memory bottleneck described previously. By having multiple concurrently executing thread contexts on one chip, a stall due to a cache miss in one thread does not stall the entire processor, leading to higher overall processor utilization. Similar to techniques using prefetch instructions described earlier, an on-chip thread can also be used to explicitly load data into the cache for other another thread [13].

Techniques that perform well on a uniprocessor may not be appropriate for a CMP. For instance, Johnson et al. found that work-sharing schemes for in-memory workloads on CMPs actually reduced performance [10]. This was because work-sharing created a bottleneck at the shared work, limiting the total parallelism. An important lesson is that in order to achieve optimal database performance on a CMP, database implementers must carefully evaluate design decisions that may limit parallelism.

Key Applications

Databases are crucial to a wide range of data storage and analysis activities. Optimizing core database operations to take advantage of all of the features and resources available on modern hardware will result in better performance, which in turn has the ability to directly improve the experience of all users of databases.

Future Directions

Adapting database operators to increased on-chip parallelism afforded by CMPs is one open problem. As the amount of on-chip parallelism increases, new bottlenecks and scaling challenges will emerge. Another area for future research includes using non-traditional architectures, such as the Cell Broadband Engine [9] and other future heterogeneous processors, for database operations. As long as computer architecture continues to evolve, architecture-conscious databases will need to adapt to new technologies.

Cross-references

► Cache-Conscious Query Processing

Recommended Reading

1. Ailamaki A., DeWitt D.J., Hill M.D., and Wood D.A. DBMSs on a modern processor: Where does time go? In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 266–277.
2. Chamberlin D.D., Astrahan M.M., Blasgen M.W., Gray J.N., King W.F., Lindsay B.G., Lorie R., Mehl J.W., Price T.G., Putzolu F., 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. Chen S., Ailamaki A., Gibbons P.B., and Mowry T.C. Improving hash join performance through prefetching. ACM Trans. Database Syst., 32(3):17, 2007.
4. Chen S., Gibbons P.B., Mowry T.C., and Valentin G. Fractal prefetching B+ trees: Optimizing both cache and disk performance. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 157–168.
5. Cieslewicz J. and Ross K.A. Adaptive aggregation on chip multiprocessors. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 339–350.
6. Cieslewicz J. and Ross K.A. Database optimizations for modern hardware. Proc. IEEE, 96(5):863–878, May 2008.
7. Graefe G. The five-minute rule twenty years later, and how flash memory changes the rules. In Proc. Workshop on Data Management on New Hardware, 2007.
8. 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, pp. 79–87.

9. Héman S., Nes N., Zukowski M., and Boncz P. Vectorized data processing on the Cell Broadband Engine. In Proc. Workshop on Data Management on New Hardware, 2007.
10. 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.
11. Lee S.-W. and Moon B. Design of flash-based DBMS: an in-page logging approach. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 55–66.
12. Ross K.A. Selection conditions in main memory. ACM Trans. Database Syst., 29:132–161, 2004.
13. Zhou J., Cieslewicz J., Ross K.A., and Shah M. Improving database performance on simultaneous multithreading processors. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 49–60.
14. Zhou J. and Ross K.A. Implementing database operations using SIMD instructions. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 145–156.

Architecture-Sensitive Database System

► Architecture-Conscious Database System

Archiving Experimental Data

Reagan W. Moore
University of California, San Diego, La Jolla, CA, USA

Synonyms

Preservation; Reference collections

Definition

The archiving of data is the process by which a project preserves both data and representation information that is needed to interpret the data.

Historical Background

An increasing fraction of science research is based on the analysis of large data collections. Experimental data are collected from measurements by sensors on repeatable experiments conducted in laboratories. Observational data are collected from sensors that measure properties of the natural environment. Simulation output files are generated by applications that run on supercomputers. The size of each of these types of research data can be massive, measured in Petabytes (thousands of Terabytes), such that there is physically not enough room to keep the data on high-speed disk file systems.

The archiving of experimental data traditionally focused on the migration of data from high-performance (more costly) storage systems to the lowest cost media that were available (originally tape-based systems). Files were manually copied from disk onto the tape archive. The IEEE Mass Storage System Reference Model version 5 defines the properties that a tape archive should provide [4]. The properties include the ability to automate retrieval of the file, manage the name space used to identify the file, and manage migration of files between tapes. While these capabilities enable the long-term management of the files, each project was required to manage independently any representation information needed to interpret the file and the meaning of the data. The OAIS model defines representation information that should be provided for preservation of data [6]. This includes information about the source of the data, descriptions of the structures present within the data file, identification of the application that can manipulate the data structure, descriptions of the meaning of the data, and identification of a knowledge community that can interpret the meaning. Archiving of experimental data is successful when the data can be retrieved, interpreted, and manipulated by the project at an unspecified future time.

Foundations

The archiving of experimental data is facilitated through the development of standards for the descriptive terms used to describe meaning and provenance, standards for describing the structures present in the file, and standards for the services that manipulate the data structures.

- Standard semantics. The terms used to describe physical phenomena are created by each discipline. The meaning of these terms evolves over time as a better understanding is developed of the physical phenomena. A preservation environment will need to support evolution of the semantic meaning, through the mapping of prior vocabulary sets to the new vocabulary. This is typically done through use of ontologies that define logical mappings between terms, spatial correlation of terms to maps, assignment of temporal coordinates to processes,

and specification of functional relationships between physical quantities.

- Standard formats. Each file is a linear string of bits, on which structure is imposed. A file format describes how to interpret the bit string into structures that can be named. Since the types of observational or experimental or simulation data vary extensively, each community chooses a standard format for specific physical phenomena. As new phenomena are measured, new data formats are created. Thus a preservation environment needs to manage either characterization of the file formats or migration of the file formats onto new standards. A promising approach to handle data format migration is based on Data Format Description Languages, that enable the description of the structure of the file using an XML syntax.
- Standard access methods. Each scientific community defines standard processing steps for manipulating the structures present within the standard data formats, using the standard semantic terms. These standard processing mechanisms can be ported on top of data grid technology to enable their use within the preservation environment [5]. An alternate approach is to port the required display application to each new operating system technology. This assumes the data can be retrieved from the preservation environment, and a copy is placed on the computer where the emulated application is executed.

The size of science data collections poses a major scalability challenge, and strongly impacts the implementation of the data archiving process. The large number of files and the large size of an individual file require support mechanisms that may not be provided by a specific storage system. The science research projects listed below use data grid technology to overcome limitations in current storage technology, and to simplify incorporation of new technology within the preservation environment. Data grids are software middleware that insulate the scientific collections from dependencies on current storage technology through the implementation of infrastructure independence [2]. The names used to describe the science data are managed by the data grid independently of the choice of storage system. This makes it possible to ensure consistent naming even when data is distributed across multiple storage systems or migrated from old technology to new technology.

Data grids provide standard operations for manipulating science data. The operations are executed at the remote storage system through interoperability mechanisms that can be applied on any choice of storage. Data grids also provide standard interfaces that enable the use of a preferred access mechanism. The result is the ability to use a particular research group's access interface to manipulate data stored on multiple types of remote storage systems.

The mechanisms provided by data grids to manage science data explicitly handle issues of scale:

- Large file size. An individual file can be multiple gigabytes in size. The time required to move such a file over a network to a storage system can be excessive. By using parallel I/O streams, the time can be decreased by multiple factors of two. Effectively, the data file is separated into multiple segments, each of which is transmitted over the network in parallel with the other segments. Large files are typically sent using four to eight parallel I/O streams.
- Large file size. The retrieval of a large file, even with parallel I/O streams may take multiple minutes. For cases where a small subset of the large file is needed, it is more efficient to filter the file at the remote storage system, and send only the desired subset back over the network. Data grids support remote manipulation of files through the execution of remote procedures at the location where the file resides.
- Large number of files. Each storage system has a maximum number of files that it is designed to handle. Large collections can exceed the number of files that can be written to a single storage system. To overcome this limitation, data grids aggregate small files into a single larger file called a container. The container is written to the storage system. The data grid maintains information that allows it to track the location in the container of each file. For example, the 2-Micron All Sky Survey [8] astronomy image collection containing 5 million images was archived on tape by aggregating the images into 147,000 containers. If the images in each container are from the same area on the sky, then retrieval of the container can result in multiple related images becoming accessible, improving bulk access to the collection.
- Large number of files. For the archived files to be useful, descriptive information is needed about

each file to support discovery. The descriptive information may be extractable from the file. However, if a discovery request is issued that requires the parsing of every file in the collection, the time to satisfy the request may be exceptionally long. Data grids manage metadata for each file in a metadata catalog. The metadata catalog is stored as tables in a relational database, enabling efficient searches. Data grids can use remote procedures to parse the descriptive metadata from the file, and bulk load the metadata into the metadata catalog.

- Large size of collections. The management of integrity across large collections may be viewed as an intractable problem. File integrity can be verified by reading each file, calculating a checksum, and then comparing the checksum with a previous value that was stored in the metadata catalog. If the file has become corrupted, the two checksums will not be equal. If a single tape drive is used to read a Petabyte collection, a sustained data transfer rate of 33 MB/s is needed to read the entire collection in a year. Thus integrity checking of large collections can require the dedication of significant hardware resources. If a problem is detected, a second copy is required to be able to repair the corruption. Data grids support the replication of data onto multiple storage systems that may be located at geographically remote locations. The geographic separation is needed to ensure recovery from natural disasters. Synchronization of replicas is done to verify integrity of the files.

Key Applications

Science disciplines are generating massive collections of experimental, observational, and simulation data. Single projects such as the Southern California Earthquake Center [7] plan to generate over 1.5 Petabytes of simulation data of seismic wave propagation from earthquakes on the San Andreas Fault. The Large Synoptic Survey Telescope (LSST) plans to capture more than 130 Petabytes of observational data [3]. The LSST project takes photographs of the sky to track near earth objects, supernovae, and micro-lensing events that can provide information on the structure of the Universe. The BaBar high-energy physics experiment has moved more than 400 Terabytes of experimental data from the Stanford Linear Accelerator in Palo Alto, California to Lyon, France for analysis by collaborating physicists [1]. In each case, the data are archived for comparison with future research results.

Future Directions

The ability to validate the trustworthiness of a digital repository is becoming an essential requirement for the archiving of experimental data. When scientific collections are archived, each community defines assessment criteria that they expect the preservation environment to maintain. The assessment criteria may be related to retention and disposition policies, or to time-dependent access controls, or to specifications of required descriptive metadata, or to required access mechanisms for data display and manipulation. An emerging requirement for scientific collections is the characterization of the management policies under which the desired collection properties are enforced.

Rule based data grids provide the mechanisms needed not only to enforce the application of the collection management policies, but also to automate the execution of data management policies. As scientific collections grow to the Petabyte size, the labor required to administer the collections can become onerous. This is driven by the use of distributed storage systems to manage the collections. Once the collection resides on multiple types of storage systems, located on multiple administrative domains at geographically remote sites, it becomes very hard to control what is happening. A network router or storage system may be taken down for maintenance, or an operational procedure may change, causing an unexpected result. Data grids provide mechanisms to recover from such problems through the use of replicas, checksums, and synchronization. A problem can be detected and repaired through an administrator-initiated action. When the number of detected problems becomes too large, the administrator is no longer able to keep up with the workload.

Rule based data grids minimize the labor required by the administrator by automating execution of management policies. Management policies are defined that control the preservation processes that are applied to the collection (e.g., validate checksum, verify presence of required metadata, implement the retention policy). Assessment criteria are specified that evaluate whether the management policies have been correctly applied. In a rule-based data grid, the preservation processes are expressed as sets of micro-services that are executed at the remote storage system. Each micro-service in turn is composed from standard operations that the data grid implements for each type of storage system. Management policies are expressed as sets of rules that control the execution of the micro-services. A rule engine is

installed at each remote storage system to ensure that the policies are enforced, no matter which access mechanism is used to interact with the data grid. The assessment criteria are mapped to queries on persistent state information that is generated after the execution of each micro-service. Such a rule based data grid is capable of monitoring its own operations and verifying the trustworthiness of the digital repository that holds the archived scientific collection [2].

Cross-references

- ▶ Data Warehouse
- ▶ Disaster Recovery
- ▶ Information Lifecycle Management
- ▶ Meta data Repository
- ▶ Provenance
- ▶ Replication

Recommended Reading

1. BaBar B meson high energy physics project. Available at: http://www.slac.stanford.edu/BFROOT/
2. Integrated Rule-Oriented Data System (iRODS). Available at: http://irods.sdsc.edu/
3. Large Synoptic Survey Telescope (LSST). Available at: http://www.lsst.org/
4. Miller S.W. Mass storage reference model special topics. In Proc. 9th IEEE Symp. on Mass Storage Systems, Monterey, CA, November 1988, pp. 3–7.
5. Moore R. Building preservation environments with data grid technology. Am. Arch., 69(1):139–158, July 2006.
6. OAIS, reference model for an open archival information system, ISO standard ISO 14721:2003. Available at: http://nost.gsfc.nasa.gov/isoas/ref_model.html
7. Southern California Earthquake Center (SCEC). Available at: http://www.scec.org/
8. 2-micron all sky survey. Available at: http://www.ipac.caltech.edu/2mass/

Armstrong Axioms

Solmaz Kolahi
University of British Columbia, Vancouver, BC, Canada

Definition

The term *Armstrong axioms* refers to the sound and complete set of inference rules or axioms, introduced by William W. Armstrong [2], that is used to test logical implication of functional dependencies.

Given a relation schema $R[U]$ and a set of functional dependencies Σ over attributes in U, a functional dependency f is logically implied by Σ, denoted by $\Sigma \models f$, if for every instance I of R satisfying all functional dependencies in Σ, I satisfies f. The set of all functional dependencies implied by Σ is called the *closure* of Σ, denoted by Σ^+.

Key Points

Armstrong axioms consist of the following three rules:

Reflexivity: If $Y \subseteq X$, then $X \rightarrow Y$.
Augmentation: If $X \rightarrow Y$, then $XZ \rightarrow YZ$.
Transitivity: If $X \rightarrow Y$ and $Y \rightarrow Z$, then $X \rightarrow Z$.

Note that in the above rules XZ refers to the union of two attribute sets X and Z. Armstrong axioms are *sound* and *complete*: a functional dependency f is derivable from a set of functional dependencies Σ by applying the axioms if and only if $\Sigma \models f$ (refer to [1] for more information).

Cross-references

- ▶ Functional Dependency
- ▶ Implication of Constraints

Recommended Reading

1. Abiteboul S., Hull R. and Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, 1995.
2. Armstrong W. Dependency structures of data base relationships. In IFIP Congress. 1974.

Array

- ▶ Redundant Array of Independent Disks (RAID)

Array Databases

- ▶ Raster Data Management and Multi-Dimensional Arrays

Association

- ▶ Abstraction
- ▶ Similarity and Ranking Operations

Association Rule Mining on Streams

Philip S. Yu[1], Yun Chi[2]
[1]IBM T.J. Watson Research Center, Yorktown Heights, NY, USA
[2]NEC Laboratories America, Cupertino, CA, USA

Definition

Let $I = \{i_1,...,i_m\}$ be a set of items. Let S be a stream of transactions in a sequential order, where each transaction is a subset of I. For an *itemset* X, which is a subset of I, a transaction T in S is said to *contain* the itemset X if $X \subseteq T$. The *support* of X is defined as the fraction of transactions in S that contain X. For a given support threshold s, X is *frequent* if the support of X is greater than or equal to $s\%$, i.e., if at least $s\%$ transactions in S contain X. For a given *confidence* threshold c, an *association rule* $X \Rightarrow Y$ holds if $X \cup Y$ is frequent and at least $c\%$ of transactions in S that contain X also contain Y. The problem of association rule mining on streams is to discover all association rules that hold in a stream of transactions.

Historical Background

In 1993, Rakesh Agrawal et al. [1] proposed the framework for association rule mining. Since this seminal work, a lot of researches have been done to improve the efficiency of association rule mining algorithms, to extend the definition of associations rule, and to apply association rule mining to other types of data such as time sequence data and structured data such as graphs. On the other hand, research on data streams started around 2000 when several data stream management systems were originated (e.g., the Brandeis AURORA project, the Cornell COUGAR project, and the Stanford STREAM project) to solve new challenges in applications such as network traffic monitoring, transaction data management, Web click streams monitoring, sensor networks, etc. Because association rule mining plays an important role in these data stream applications (e.g., the motivation in the first association rule paper [1] is to mine patterns from retail store transaction data), along with the development of data stream management systems, developing association rule mining algorithms for data streams has become an important research topic.

Foundations

Two Sub-problems

Algorithms for association rule mining usually consist of two steps. The first step is to discover frequent itemsets. In this step, all frequent itemsets that meet the support threshold are discovered. The second step is to derive association rules. In this step, based on the frequent itemsets discovered in the first step, the association rules that meet the confidence criterion are derived. Because the second step, deriving association rules, can be solved efficiently in a straightforward manner, most of researches focus mainly on the first step, i.e., how to efficiently discover all frequent itemsets in data streams. Therefore, in the rest of this article, the focus will be on frequent itemset mining in data streams.

Key Challenges

Frequent itemset mining in general is already a challenging problem. For example, due to combinatorial explosion, there may be huge number of frequent itemsets, and a main challenge is how to efficiently enumerate, discover, and store frequent itemsets. Data streams, because of their unique features, have further posed many new challenges to frequent itemset mining. Some of these new challenges are described as the following.

Single access of data In data streams, data are arriving continuously with high speed and in large volume. As a consequence, in many cases it is impractical to store all data in persistent media and in other cases, it is too expensive to (randomly) access data multiple times. The challenge is to discover frequent itemsets while the data can only be assessed once.

Unbounded data Another feature of data streams is that data are unbounded. In comparison, storage that can be used to discover or maintain the frequent itemsets is limited. Another consequence of unbounded data is that whether an itemset is frequent depends on time. The challenge is to use limited storage to discover dynamic frequent itemsets from unbounded data.

Real-time response Because data stream applications are usually time-critical, there are requirements on response time. For some restricted scenarios, algorithms that are slower than the data arriving rate are useless. The challenge is therefore to efficiently mine frequent itemsets in real time.

Data Models

Compared with finite data in traditional databases, data streams are unbounded and the number of transactions increase with time. Due to these characteristics, different data models have been proposed in different mining algorithms.

The first data model is an accumulative model. In this model, all data in the range from the beginning to the current time are considered in an equal fashion. Therefore, frequent itemsets are defined on the accumulated data where new data are appended continuously as time grows.

The second data model is based on a sliding window. That is, although the whole data stream is unbounded, the frequent itemsets are defined based on the most recent data that fall within a temporal sliding window whose ending time is the current time. One justification for such a sliding-window model is that due to concept drifts, the data distribution in streams is usually changing with time, and very often people are interested in the most recent patterns.

The third data model falls in between the first two models – while all data are considered in frequent itemset mining, they are weighted differently according to a predefined weighting function. A very commonly used weighting function is the exponentially decaying function, that is, for a transaction at time τ, its weight is $\alpha(\tau) = \exp(\tau - t)$, where t is the current time.

Algorithm Types

Based on the mining results, existing frequent itemsets mining algorithms on data streams can be roughly divided into two categories: the exact mining algorithms and the approximate mining algorithms.

Exact mining algorithms provide as results all the frequent itemsets in the data streams together with their accurate supports. Usually the focus of exact algorithms is to efficiently update frequent itemsets when new transactions arrive and (in the sliding-window data model) when old transactions expire.

Approximate mining algorithms, on the other hand, focus more on finite memory usage and single access of data, at the cost of the accuracy of the mining results. Approximate algorithms target low false positive rate, zero, or low false negative rate, and tight error bounds on the estimation for the supports of the frequent itemsets.

Representative Algorithms

Cheung et al. [6,7] proposed algorithms *FUP* and FUP_2 for incrementally updating frequent itemsets. Thomas et al. [13] presented a similar algorithm. Both Cheung's and Thomas's algorithms assume batch updates and take advantage of the relationship between the original database (*DB*) and the incrementally changed transactions (*db*). *FUP* is similar to the well-known Apriori algorithm [1], which is a multiple-step algorithm. The key observation of *FUP* is that by adding *db* to *DB*, some previously frequent itemsets will remain frequent and some previously infrequent itemsets will become frequent (these itemsets are called *winners*); at the same time, some previously frequent itemsets will become infrequent (these itemsets are called *losers*). The key technique of *FUP* is to use information in *db* to filter out some winners and losers, and therefore reduce the size of candidate set in the Apriori algorithm. Because the performance of the Apriori algorithm relies heavily on the size of candidate set, *FUP* improves the performance of Apriori greatly. FUP_2 extended *FUP* by allowing deleting old transactions from a database as well. Therefore *FUP* is restricted to the accumulative data model while FUP_2 can be used on the sliding-window data model as well. The algorithm proposed by Thomas et al. is similar to FUP_2 except that in addition to frequent itemsets, a negative border is maintained. In the algorithm, the frequent itemsets in *db* are mined first. At the same time, the counts of frequent itemsets (and itemsets on the negative border) in *DB* are updated. Then based on the change of the frequent itemsets in *DB*, the negative border in *DB*, and the frequent itemsets in *db*, the frequent itemsets in the updated database are computed with a possible scan of the updated database. Because the updated database is scanned at most once, Thomas's algorithm has very good performance. Thomas's algorithm can be used for both the accumulative and the sliding-window data models. In addition, *FUP*, FUP_2, and Thomas's algorithm all fall into the category of exact mining algorithms.

Veloso et al. [14] proposed an algorithm *ZIGZAG* for mining frequent itemsets in evolving databases. Later, Otey et al. [11] extended *ZIGZAG* into parallel and distributed algorithms. *ZIGZAG* is similar to Cheung's and Thomas's algorithms in that it achieves

its speedup by using the relationship between *DB* and *db*. However, *ZIGZAG* has many distinct features. First, *ZIGZAG* mainly used *db* to speedup the support counting of frequent itemsets in the updated database and it does not discover the frequent itemsets in *db* itself. As a result, for a given minimum support, *ZIGZAG* can handle batch update with arbitrary block size. Second, *ZIGZAG* adapts the techniques proposed in the *GENMAX* algorithm [9] and in each update only maintains *maximal* frequent itemsets. Because the information on maximal frequent itemsets and their supports is not enough to generate association rules (because the support information of some non-maximal frequent itemsets may be missing), a second step is used in *ZIGZAG* in which the updated database is scanned to discover all frequent itemsets and their supports.

Chi et al. [5] developed an algorithm, *Moment*, to mine *closed* frequent itemsets over data stream sliding windows. In this work the authors introduced a compact data structure, the *closed enumeration tree* (CET), to maintain a dynamically selected set of itemsets over a sliding window. The selected itemsets contain a boundary between closed frequent itemsets and the rest of the itemsets. Concept drifts in a data stream are reflected by boundary movements in the CET, and can be efficiently captured. Both *ZIGZAG* and *Moment* are exact mining algorithms that use the sliding-window data model.

Charikar et al. [3] presented a one-pass algorithm, *Count Sketch*, that returns most frequent *items* whose frequencies satisfy a threshold with high probabilities. Manku et al. [10] developed a randomized algorithm, the *Sticky Sampling* algorithm, and a deterministic algorithm, the *Lossy Counting* algorithm, for maintaining frequent *items* over a data stream where for a given time *t*, the frequent items are defined over the *entire* data stream up to *t*. The algorithms guarantee no false negative and a bound on the error of estimated frequency (the guarantees are in a probabilistic sense for the randomized algorithm). The *Lossy Counting* algorithm is extended to handle frequent *itemsets*, where a trie is used to maintain all frequent itemsets and the trie is updated by batches of transactions in the data stream. The algorithms of Manku et al. strive for a tunable compromise between memory usage and error bounds. *Count Sketch*, *Sticky Sampling*, and *Lossy Counting* are all approximate mining algorithms that use the accumulative data model.

Teng et al. [12] presented an algorithm, *FTP-DS*, that mines frequent temporal patterns from data streams of itemsets. *FTP-DS* is an approximate mining algorithm that uses the sliding-window data model. Chang et al. [2] presented an algorithm, *estDec*, that mines recent frequent itemsets where the frequency is defined by an aging function. Giannella et al. [8] proposed an approximate algorithm for mining frequent itemsets in data streams during arbitrary time intervals. An in-memory data structure, *FP-stream*, is used to store and update historic information about frequent itemsets and their frequency over time and an aging function is used to update the entries so that more recent entries are weighted more. Both *estDec* and Giannella's algorithm are approximate mining algorithms on weighted transactions. However, Giannella's algorithm is a little different in that it can provide different error levels for data at multiple time granularities.

The above representative algorithms are summarized in Table 1. For a more detailed survey on algorithms for frequent itemset mining over data streams, refer to a recent survey by Cheng et al. [4].

Key Applications

For most data stream applications, there are needs for mining frequent patterns and association rules from data streams. Some key applications in various areas are listed in the following.

Performance monitoring Monitor network traffic and performance, detect abnormality and intrusion

Association Rule Mining on Streams. Table 1. The categorization of the representative algorithms according to their data models and algorithm types

	Data Model		
	Accumulative	Sliding-window	Weighted
Exact Mining Algorithm	*FUP* [2]	*FUP*$_2$ [3] Thomas's [4] *ZIGZAG* [5, 6] *Moment* [8	
Algorithm Mining Algorithm	*Count Sketch* [9] *Sticky Sampline* [10] *Lossy Counting* [10]	*FTP-DS* [11	*estDec* [12] Giannella's [13]

Transaction monitoring Monitor transactions in retail stores, ATM machines, and financial markets
Log record mining Mine patterns from telecommunication calling records, Web server log, etc.
Sensor network mining Mine patterns in streams coming from sensor networks or surveillance cameras

Experimental Results

In general, for each of the presented algorithms, there are supporting experimental studies in the corresponding references. The commonly compared performance metrics include memory usage, speed of the mining process, and (for the approximate algorithms) errors such as false positive rate, false negative rate, and support errors for the frequent itemsets.

Data Sets

A Linux version of the synthetic data generator originally developed by Agrawal et al. [1] is available at http://miles.cnuce.cnr.it/~palmeri/datam/DCI/datasets.php

Some other synthetic and real-life data sets are available from the Frequent Itemset Mining Dataset Repository at http://fimi.cs.helsinki.fi/data/

Furthermore, pointers to some related data sets are available at the KDnuggets Web site http://www.kdnuggets.com/datasets/

Cross-references

- ▶ Approximation of Frequent Itemsets
- ▶ Association Rule Mining
- ▶ Change Detection on Streams
- ▶ Closed Itemset Mining and Nonredundant Association Rule Mining
- ▶ Continuous Queries in Sensor Networks
- ▶ Data Aggregation in Sensor Networks
- ▶ Data Estimation in Sensor Networks
- ▶ Data Mining
- ▶ Data Sketch/Synopsis
- ▶ Data Streams
- ▶ Frequent Items on Streams
- ▶ Frequent Itemsets and Association Rules
- ▶ Incremental Computation of Queries
- ▶ Internet and Web Transactions
- ▶ One-Pass Algorithm
- ▶ Pattern-Growth Methods
- ▶ Randomization Methods
- ▶ Real-Time Transactions
- ▶ Sensor Network
- ▶ Stream Mining
- ▶ Stream Models
- ▶ Streaming Applications
- ▶ Tries
- ▶ Web Services

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
2. Chang J.H. and Lee W.S. Finding recent frequent itemsets adaptively over online data streams. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 487–492.
3. Charikar M., Chen K., and Farach-Colton M. Finding frequent items in data streams. In Proc. 29th Int. Colloquium on Automata, Languages and Programming, 2002, pp. 693–703.
4. Cheng J., Ke Y., and Ng W. A survey on algorithms for mining frequent itemsets over data streams. Knowledge and Int. Syst., 16(1):1–27, 2008.
5. Chi Y., Wang H., Yu P.S., and Muntz R.R. Catch the moment: maintaining closed frequent itemsets in a data stream sliding window. Knowl. Inf. Syst., 10(3):265–294, 2006.
6. Cheung D.W., Han J., Ng V., and Wong C.Y. Maintenance of discovered association rules in large databases: an incremental updating technique. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 106–114.
7. Cheung D.W., Lee S.D., and Kao B. A general incremental technique for maintaining discovered association rules. In Proc. 5th Int. Conf. on Database Systems for Advanced Applications, 1997, pp. 185–194.
8. Giannella C., Han J., Pei J., Yan X., and Yu P.S. Mining frequent patterns in data streams at multiple time granularities. In H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.). Data Mining: Next Generation Challenges and Future Directions. AAAI, 2004.
9. Gouda K. and Zaki M.J. Efficiently mining maximal frequent itemsets. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 163–170.
10. 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.
11. Otey M.E., Parthasarathy S., Wang C., Veloso A., and Meira W. Parallel and distributed methods for incremental frequent itemset mining. IEEE Trans. Syst. Man Cybern. B, 34(6):2439–2450, 2004.
12. Teng W.-G., Chen M.-S., and Yu P.S. A regression-based temporal Pattern Mining Scheme for Data Streams. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 98–104.
13. Thomas S., Bodagala S., Alsabti K., and Ranka S. An efficient algorithm for the incremental updation of association rules in large databases. In Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining, 1997, pp. 263–266.
14. Veloso A., Meira Jr. W., and de Carvalho M., Pôssas B., Parthasarathy S., and Zaki M.J. Mining frequent itemsets in evolving databases. In Proc. SIAM International Conference on Data Mining, 2002.

Association Rule Visualization

► Visual Association Rules

Association Rules

Jian Pei
Simon Fraser University, Burnaby, BC, Canada

Definition

Let $\mathcal{I}$ be a set of *items*, where each item is a literal. A *transaction* $T \subseteq \mathcal{I}$ is a subset of $\mathcal{I}$. Association rules are defined on a set of transactions $\mathcal{T}$.

An association rule R is in the form of $X \rightarrow Y$, where X and Y are two sets of items, that is, $X, Y \subseteq \mathcal{I}$. R is associated with two measures, the *support* $sup(R)$ and the confidence $conf(R)$. The support $sup(R)$ is the probability that X appears in a transaction in $\mathcal{T}$. The confidence $conf(R)$ is the conditional probability that when X appears in a transaction, Y also appears.

Historical Background

The concept of association rules were firstly proposed by Agrawal et al. [1] for market basket analysis. A well known illustrative example of association rules is "Diaper → Beer" which can be explained by the fact that, when dads buy diapers for their babies, they also buy beer at the same time for their weekends game watching.

Apriori, an efficient algorithm for mining association rules, was developed by Agrawal and Srikant [3], while the similar idea was explored by Mannila et al. [13]. There have been many studies trying to improve the efficiency of Apriori (refer to entry "Apriori property and breadth-first search algorithms"). A pattern-growth approach for mining frequent itemsets without candidate generation was developed by Han et al. [8], and has been further improved by many studies since 2001 (see entry "Pattern-growth methods").

To remove redundancy in association rules, Pasquier et al. [15] proposed the notion of frequent closed itemsets using formal concept analysis. Several efficient algorithms have been developed (refer to entry "Closed itemset mining and non-redundant association rule mining").

Association rules have been extended in several ways, such as sequential patterns and sequential association rules (refer to entry "Sequential patterns"), spatial association rules [9], cyclic association rules [14], negative association rules [19], intertransaction association rules [12], multilevel generalized association rules [7,20], and quantitative association rules (refer to entry "Quantitative association rules").

In addition to support and confidence, some other interestingness measures for association rules were explored, such as [5,18].

Foundations

Let $\mathcal{I}$ be a set of *items*, where each item is a literal. An *itemset* X is a subset of items, that is, $X \subseteq \mathcal{I}$. A *transaction* $T \subseteq \mathcal{I}$ consists of a *transaction id* and an itemset. A *transaction database* $\mathcal{T}$ is a multiset of transactions.

An association rule R is in the form of $X \rightarrow Y$, where X and Y are two itemsets, that is, $X, Y \subseteq \mathcal{I}$. R is associated with two measures, the *support* $sup(R)$ and the confidence $conf(R)$. The support $sup(R)$, given by $sup(R) = Pr(X)$, is the probability that X appears in a transaction in $\mathcal{T}$. The confidence $conf(R)$, given by $conf(R) = \frac{sup(X \cup Y)}{sup(X)} = Pr(Y|X)$, is the conditional probability that when X appears in a transaction, Y also appears.

Given a transaction database $\mathcal{T}$, a minimum support threshold *minsup*, and a minimum confidence threshold *minconf*, the problem of *association rule mining* is to find the complete set of association rules whose supports are at least *minsup* and whose confidences are at least *minconf*.

Association rules can be mined in two steps. In the first step, the complete set of frequent itemsets are identified. An itemset is called *frequent* if $Pr(X) \geq minsup$. In the second step, frequent itemsets are used to generate association rules.

More often than not, to make association rule mining interesting, a user may specify a *minimum support threshold min_sup* and a *minimum confidence threshold min_conf*. Then, only the association rules whose supports and confidence pass those thresholds, respectively, should be returned. Alternatively, in some situations, a user may want to find the top-k association rules with the largest support and/or confidence. Some other kinds of constraints can also be specified. Such thresholds and constraints may be used by some association rule mining methods to speed up the mining procedure.

Key Applications

Association rules have been extensively mined and used in many applications. For example, mining association rules about customers' market baskets helps to identify the products that customers like to purchase together, or some products that may trigger the purchases of some other products. Such information can help business in one way or another. For example, knowing that the purchase of diapers may potentially lead to the purchase of beer, a store can put beer beside diapers so that the sales of beer can be boosted.

Association rules are also mined on biological data and clinic data. For example, mining the association rules among symptoms and disease can help to diagnose diseases.

An important application of association rules is to construct classifiers using association rules. Technically, association rules among features and the target class labels can be mined and the rules can be used to make prediction on cases with unknown class labels. Research has found that associative classifiers, classifiers using association rules, are accurate and highly understandable in a few applications such as those with many features [6,10,11].

Cross-references

- ► Approximation of Frequent Itemsets
- ► Apriori Property and Breadth-First Search Algorithms
- ► Associative Classifiers
- ► Closed Itemset Mining and Non-Redundant Association Rule Mining
- ► Data Mining
- ► Emerging Patterns
- ► Frequent Itemset Mining with Constraints
- ► Frequent Itemsets and Association Rules
- ► Pattern-Growth Methods
- ► Quantitative Association Rules
- ► Sequential Patterns
- ► Sequential Patterns with Constraints

Recommended Reading

1. Agrawal R., Imielinski T., and Swami A. Mining association rules between sets of items in large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993, pp. 207–216.
2. Agrawal R., Mannila H., Srikant R., Toivonen H., and Verkamo A.I. Fast discovery of association rules. In Advances in Knowledge Discovery and Data Mining, U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy (eds.). AAAI/MIT, Menlo Park, CA/Cambridge, MA, 1996, pp. 307–328.
3. Agrawal R. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
4. Agrawal R. and Srikant R. Mining sequential patterns. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 3–14.
5. Brin S., Motwani R., and Silverstein C. Beyond market basket: generalizing association rules to correlations. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1997, pp. 265–276.
6. Dong G. and Li J. Efficient mining of emerging patterns: discovering trends and differences. In Proc. 5th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 1999, pp. 43–52.
7. Han J. and Fu Y. Discovery of multiple-level association rules from large databases. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 420–431.
8. Han J., Pei J., and Yin Y. Mining frequent patterns without candidate generation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 1–12.
9. Koperski K. and Han J. Discovery of spatial association rules in geographic information databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 47–66.
10. Li W., Han J., and pei J. CMAR: accurate and efficient classification based on multiple class-association rules. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 369–376.
11. Liu B., Hsu W., and Ma Y. Discovering the set of fundamental rule changes. In Proc. 7th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2001, pp. 335–340.
12. Lu H., Han J., and Feng L. Stock movement and n-dimensional inter-transaction association rules. In Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 1998, pp. 1201–1217.
13. Mannila H., Toivonen H., and Verkamo A.I. Efficient algorithms for discovering association rules. In Proc. AAAI 1994 Workshop Knowledge Discovery in Databases, 1994, pp. 181–192.
14. Özden B., Ramaswamy S., and Silberschatz A. Cyclic association rules. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 412–421.
15. Pasquier N., Bastide Y., Taouil R., and Lakhal L. Discovering frequent closed itemsets for association rules. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 398–416.
16. Pei J., Han J., Lu H., Nishio S., Tang S., and Yang D. H-Mine: hyper-structure mining of frequent patterns in large databases. In Proc. 2001 IEEE Int. Conf. on Data Mining, 2001, pp. 441–448.
17. Pei J., Han J., and Mao R. CLOSET: an efficient algorithm for mining frequent closed itemsets. In Proc. ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp. 11–20.
18. Piatetsky-Shapiro G. Discovery, analysis, and presentation of strong rules. In Knowledge Discovery in Databases, G. Piatetsky-Shapiro and W. Frawley (eds.). AAAI/MIT, Menlo Park, CA/Cambridge, MA, 1991, pp. 229–238.
19. Savasere A., Omiecinski E., and Navathe S. Mining for strong negative associations in a large database of customer transactions. In Proc. 14th Int. Conf. on Data Engineering, 1998, pp. 494–502.
20. Srikant R. and Agrawal R. Mining generalized association rules. In Proc. 21th Int. Conf. on Very Large Data Bases, 1995, pp. 407–419.

Associative Classification

► Classification by Association Rule Analysis

Asymmetric Encryption

Ninghui Li
Purdue University, West Lafayette, IN, USA

Synonyms

Public-key encryption

Definition

Asymmetric encryption, also known as public-key encryption, is a form of data encryption where the encryption key (also called the public key) and the corresponding decryption key (also called the private key) are different. A message encrypted with the public key can be decrypted only with the corresponding private key. The public key and the private key are related mathematically, but it is computationally infeasible to derive the private key from the public key. Therefore, a recipient could distribute the public key widely. Anyone can use the public key to encrypt messages for the recipient and only the recipient can decrypt them.

Key Points

A public-key encryption algorithm requires a trapdoor one-way function, i.e., a function that is easy to compute but hard to invert unless one knows some secret trapdoor (i.e., the private key). Existing public-key encryption algorithms are based on computational problems in number theory.

The most well-known public-key encryption algorithms include RSA and El Gamal. RSA uses exponentiation modulo a product of two large primes to encrypt and decrypt, and its security is connected to the presumed difficulty of factoring large integers. The El Gamal cryptosystem relies on the difficulty of the discrete logarithm problem. The introduction of elliptic curve cryptography in the mid 1980s has yielded a new family of analogous public-key algorithms. Elliptic curves appear to provide a more efficient way to leverage the discrete logarithm problem, particularly with respect to key size.

Cross-references

► Data Encryption
► Symmetric Encryption

Recommended Reading

1. Diffie W. and Hellman M.E. New directions in cryptography. IEEE Trans. Inf. Theory, 22:644–654, 1976.
2. El Gamal T. A public key cryptosystem and a signature scheme based on discrete logarithms. In Advances in Cryptology: Proc. CRYPTO '84, LNCS, vol. 196, Springer, 1985, pp. 10–18
3. Rivest R.L., Shamir A., and Adleman L.M. A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, 21:120–126, 1978.

ATA

► Storage Protocols

Atelic Data

Vijay Khatri[1], Richard T. Snodgrass[2], Paolo Terenziani[3]
[1]Indiana University, Bloomington, IN, USA
[2]University of Arizona, Tucson, AZ, USA
[3]University of Turin, Turin, Italy

Synonyms

Snapshot data; Point-based temporal data

Definition

Atelic data is temporal data describing facts that do not involve a goal or culmination. In ancient Greek, *telos* means "goal," and α is used as prefix to denote negation. In the context of temporal databases, atelic data is that data for which both *upward* and *downward* (temporal) *inheritance* holds. Specifically,

- *Downward inheritance*. The *downward inheritance* property implies that one can infer from temporal data *d* that holds at valid time *t* (where t is a time period) that *d* holds in any sub-period (and sub-point) of *t*.
- *Upward inheritance*. The *u pward inheritance* property implies that one can infer from temporal data *d* that holds at two consecutive or overlapping time

periods t_1 and t_2 that d holds in the union time period $t_1 \cup t_2$.

Atelic data is differentiated from telic data, in which neither upward nor downward inheritance holds.

Key Points

Starting from Aristotle [1], researchers in areas such as philosophy, linguistics, cognitive science and computer science have noticed that different types of facts can be distinguished according to their temporal behavior. Specifically, since atelic facts do not have any specific goal or culmination, they can be seen as "temporally homogeneous" facts, so that both upward and downward inheritance holds for them. For example, the fact that an employee (say, John) works for a company (say, ACME) would be considered atelic because of lack of goal or accomplishment. As a consequence, if John has worked for ACME from January 20, 2007 to September 23, 2007, it can be correctly inferred that John was working for ACME in May 2007 (or at any specific time point in May; therefore, downward inheritance does hold); furthermore, from the additional fact that John has also worked for ACME from September 23, 2007 to February 2, 2008, it can be correctly inferred that John worked for ACME from January 20, 2007 to February 2, 2008 (therefore, upward inheritance does hold).

In Aristotle's categorization, all possible facts are divided into two categories, telic and atelic; a telic fact, e.g., "John built a house" has goal or culmination [1].

Since both upward and downward inheritance properties hold for atelic data, such data supports the conventional "snapshot-by-snapshot" (i.e., point-based) viewpoint: the intended semantics of a temporal database is the set of conventional (atemporal) databases holding at each time point. As a consequence, most approaches to temporal databases support (only) atelic facts even if, in several cases, the representation allows, as a syntactic sugar, the use of periods to denote convex sets of time points. An integrated temporal database model that supports both telic and atelic data semantics has been developed [2].

Cross-references

- ▶ Period-Stamped Temporal Models
- ▶ Point-Stamped Temporal Models
- ▶ Telic Distinction in Temporal Databases

Recommended Reading

1. Aristotle. The Categories. On Interpretation. Prior Analytics. Harvard University Press, Cambridge, MA, 2002.
2. Terenziani P. and Snodgrass R.T. Reconciling point-based and interval-based semantics in temporal relational databases: a proper treatment of the telic/atelic distinction. IEEE Trans. Knowl. Data Eng., 16(4):540–551, 2004.

Atomic Event

Jonas Mellin, Mikael Berndtsson
University of Skövde, Skövde, Sweden

Synonyms

Primitive event

Definition

An atomic event is considered to be indivisible and instantaneous.

Key Points

If an event is non-instantaneous, then it is possible to divide into a beginning of this event (an initiator) and an ending of this event (a terminator). Therefore, atomic events must be instantaneous. In active database literature, the concept primitive event is typically used instead of atomic event. The major reason is probably that system primitives and application primitives are not distinguished.

Cross-references

- ▶ Composite Event
- ▶ Event
- ▶ Event Detection
- ▶ Event Specification

Atomicity

Gerhard Weikum
Max-Planck Institute for Informatics, Saarbruecken, Germany

Definition

The *atomicity of actions* on a database is a fundamental guarantee that database systems provide to application

programs. Whatever state modifications an atomic action may perform are guaranteed to be executed in an *all-or-nothing* manner: either all state changes caused by the action will be installed in the database or none. This property is important in the potential presence of failures that could interrupt the atomic action. The database system prepares itself for this case by *logging* state modifications and providing automated *recovery* as part of the failure handling or system restart. These implementation aspects are transparent to the application program and are thus a major relief for the programs' failure handling and boost the application development productivity.

Historical Background

Since the early 1970s (or even earlier), transaction processing systems for airline reservations and debit/credit banking had means for recovery and concurrency control that were similar to atomic actions. However, these implementation techniques were hardly documented in publicly available literature and still far from a principled, universal solution. The major credit for the modern concept of atomicity (and transactions) belongs to the 1998 Turing Award winner Jim Gray [6–8]. Closely related notions of atomic actions have been proposed by other authors at around the same time, including [11–13], which in turn were inspired by the informal work on "spheres of control" by Bjork and Davies [3,4].

Foundations

As an example for the importance of atomic actions, consider a sequence of steps that read and write two bank-account records, x and y, in order to transfer some amount of money from account x to account y:

$$R(x)W(x)R(y)W(y).$$

If there is a system failure after writing x but before writing y, the underlying database becomes inconsistent, with the transferred money seemingly lost in "mid-flight." Even worse, the application program may not even know if this problem actually occurred or not (the system may have succeeded in writing y just before it crashed but could not send a return code anymore). If, on the other hand, the database system executes the entire step sequence as an atomic action, the failure handling for the application program becomes much simpler as it can always restart on a clean, consistent database.

In database systems the atomic actions themselves can be flexibly defined by the application programs, by demarcating the begin and end of a *transaction* with explicit interface calls. For example, an entire sequence of SQL command invocations can be made atomic. By default, usually every individual SQL operation is guaranteed to be atomic. These guarantees are part of the *transactional ACID properites*: atomicity, consistency-preservation, isolation, durability [8]. In some scientific communities outside of database systems research, atomicity is meant to include the *isolation* guarantee. Atomic actions are then (alternatively) defined to be state-modification sequences whose effects are ensured (by the underlying run-time environment) to be equivalent to *indivisible actions* with all effect appearing to be instantaneous upon the completion of the entire sequence. In the presence of concurrent accesses to the same shared data, this combined atomicity/isolation property provides the illusion, despite the fact that in reality accesses by different programs are interleaved. Defining this principle in formal terms leads to the concept of *serializability* and methods for *concurrency control* as part of the database (or other run-time) system [2,6,14].

As an example for the importance of isolation (as an additional property of atomic actions), consider an extended variant of the earlier example. Assume to the the money-transfer process – now viewed as a transaction T1 – , a second transaction T2 reads both bank-account records x and y in order to analyze financial portfolios and perform some kind of risk assessment. The following concurrent execution, with time proceeding from left to right, could be possible:

T1: R(x) W(x) R(y) W(y)

T2: R(x) R(y)

With this step interleaving, transaction T2 would see an inconsistent database, namely, the state of x after money is withdrawn from x and the state of y before money is deposited there. This may lead to a distorted analysis and false conclusions in the decision making of a financial broker. Running both T1 and T2 as atomic and isolated transactions would prevent this particular interleaving and guarantees that only such executions are allowed that are provably equivalent to a sequential execution where such an anomaly is impossible.

The atomicity guarantee includes all "side effects" of an action as far as the database state is concerned. For example, the effects of a database trigger are covered by the guarantee, but effects outside of the database such as sending a message are outside the scope of the database system guarantees. Modern application servers and message brokers, on the other hand, may provide such guarantees about messages (e.g., atomic multicasts) and application state (e.g., specific program variables) beyond the database. A traditional implementation technique to this end is to support failure-resilient queues. Modern database systems have integrated such message queues and application state management and extend their atomic actions to them, providing more comprehensive *application recovery*. A new research trend in programming languages is to provide atomicity guarantees to arbitrary programs, not just database applications, in order to simplify exception handling and generally ease programmers' work. For example, method invocations in an object-oriented language could be made atomic by means of an underlying *transactional memory* as part of the language's run-time system (and possibly even hardware architecture).

The atomicity concept simplifies failure handling at the application program level, but it does not mask failures. Rather a typical approach is that the program notices the failing of an atomic action by a corresponding system return code (for the atomic action invocation itself, for the end-of-action demarcation call, or upon the next interaction with the database system if the previous call simply timed out without any response), and then has to retry the action. This paradigm still requires explicit coding for the retrying, and this may require special care about non-idempotent effects or additional system guarantees and state testing for ensuring *idempotence*. Some advanced methods for application recovery can automate these re-trials and testing for non-idempotence, thus strengthening the all-or-nothing guarantee for atomicity into an *exactly-once execution* guarantee with complete failure masking [1].

On the other hand, for some data-intensive applications outside of database systems, atomicity may be an overly strong property if applied to entire processes; this holds particularly for long-lived workflows and cooperative work. Although these applications still benefit from atomic actions for smaller-grained operations, additional forms of *relaxed atomicity* or extended atomicity would be desirable. The database research community has developed a variety of such models, most notably, the model of open nested transactions and the ACTA framework [5,9].

Future Directions

Atomicity is a ground-breaking, fundamental contribution that first emerged in database systems, but is increasingly pursued also by other research communities like programming languages, operating systems, dependable system design, and also formal reasoning and program verification [10]. There are several strategic reasons for this growing interest and extended application of atomic actions:

- Web Services, long-running workflows across organizations, large scale peer-to-peer platforms, and ambient-intelligence environments with huge numbers of mobile and embedded sensor/actor devices critically need support for handling or even masking concurrency and component failures, and may mandate rethinking the traditional atomicity concept.
- There is a proliferation of open systems where applications are constructed from pre-existing components. The components and their configurations are not known in advance and they can change on the fly. Thus, it is crucial that atomicity properties of components are composable and that one can predict and reason about the behavior of the composite system.
- Modern applications and languages like Java lead millions of developers into concurrent programming. This is a drastic change from the classical situation where only a few hundred "five-star wizard" system programmers and a few thousand programmers working in scientific computing on parallel supercomputers would have to cope with the inherently complex issues of concurrency and advanced failure handling.
- On an even broader scale, the drastically increasing complexity of the new and anticipated applications will require enormous efforts and care towards dependable systems or it may lead into a major "dependability crisis." Atomicity is an elegant basic asset to build on in the design, implementation, and composition of complex systems and the reasoning about system behavior and guaranteed properties.

Cross-references

- ▶ ACID Properties
- ▶ Concurrency Control
- ▶ Open Nested Transactions
- ▶ Software Transactional Memory
- ▶ System Recovery
- ▶ Transaction

Recommended Reading

1. Barga R.S. and Lomet D.B. German Shegalov, Gerhard Weikum: Recovery guarantees for Internet applications. ACM Trans. Internet Technol., 4(3):289–328, 2004.
2. Bernstein P.A. and Hadzilacos V. Nathan Goodman: Concurrency Control and Recovery in Database Systems. Addison-Wesley, MS, 1987.
3. Bjork L.A. Recovery scenario for a DB/DC system. In Proc. 1st ACM Annual Conference, 1973, pp. 142–146.
4. Davies C.T. Recovery semantics for a DB/DC system. In Proc. First ACM Annual Conference, 1973, pp. 136–141.
5. Elmagarmid A.K. (ed.). Database Transaction Models for Advanced Applications. Morgan Kaufmann, San Fransisco, CA, 1992.
6. Eswaran K.P., Gray J., Lorie R.A., and Traiger I.L. The Notions of consistency and predicate locks in a database system. Commun. ACM 19(11):624–633, 1976.
7. Gray J. Notes on Database Operating Systems. In Operating Systems – An Advanced Course, Springer, London, UK, 1978.
8. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Fransisco, CA, 1993.
9. Jajodia S. and Kerschberg L. (eds.). Advanced Transaction Models and Architectures Kluwer, Noewell, MA, 1997.
10. Jones C.B., Lomet D.B., Romanovsky A.B., Weikum G., Fekete A., Gaudel M.-C., Korth H.F., Rogério de Lemos, J., Moss E.B., Rajwar R., Ramamritham K., Randell B., and Rodrigues L. The atomic manifesto: a story in four quarks. ACM SIGMOD Rec., 34(1):63–69, 2005.
11. Lampson B. Atomic Transactions. In: Distributed Systems – \Architecture and Implementation, Springer, New York, NJ, 1981.
12. Lomet D.B. Process structuring, synchronization, and recovery using atomic actions. In Proc. ACM Conf. on Language Design for Reliable Software, 1977, pp. 128–137.
13. Randell B. System structure for software fault-tolerance. IEEE Trans. Softw. Eng., 1(2):221–232, 1975.
14. Weikum G. and Vossen G. Transactional Information Systems: Theory, Algorithms, and the Practice of Concurrency Control and Recovery. Morgan Kaufmann, San Fransisco, CA, 2002.

Attribute or Value Correspondence

- ▶ Schema Matching

Audible Sound

- ▶ Audio

Audio

Lie Lu[1], Alan Hanjalic[2]
[1]Microsoft Research Asia, Beijing, China
[2]Delft University of Technology, Delft, The Netherlands

Synonyms

Audible sound

Definition

Audio refers to audible sound – the sound perceivable by the human hearing system, or the sound of a frequency belonging to the *audible frequency range* (20-20,000 Hz). Audio can be generated from various sources and perceived as speech, music, voices, noise, or any combinations of these. The perception of an audible sound starts by the sound pressure waves hitting the eardrum of the outer ear. The generated vibrations are transmitted to the cochlea of the inner ear to produce mechanical displacements along the basilar membrane. These displacements are further transduced into electrical activity along the auditory nerve fibers, and finally "analyzed" and "understood" in the central auditory system [4,7].

Historical Background

The step from the fundamental definition of audio towards the concept of *audio signal* can be seen as a step towards the birth of the modern consumer electronics. An audio signal is a signal that contains audio information in the audible frequency range. The technology for generating, processing, recording, broadcasting and retrieving audio signals, first *analog* and later on *digital* ones, has rapidly grown for over a century, from the pioneering radio broadcasting and telephony systems to advanced mobile communication infrastructures, music players, speech recognition and synthesis tools, and audio content analysis, indexing and retrieval solutions. This growth may have been initiated by the research in the field of signal

processing, but it has been maintained and has continuously gained in strength through an extensive interdisciplinary effort involving signal processing, information theory, human-computer interaction, psychoacoustics, psychology, natural language processing, network and wireless technology, and information retrieval.

Foundations

Digital Audio

An audio signal is an analog signal, which can be represented as a one-dimensional function $x(t)$, where t is a continuous variable representing time. To facilitate storage and processing of such signals in computers, they can be transformed into *digital signals* by *sampling* and *quantization*.

Sampling is the process in which one audio signal value (*sample*) is taken for each time interval (*sampling period*) T. This results in a *discrete audio* signal $x(n) = x(nT)$, where n is a numeric sequence. The sampling period T determines the *sampling frequency* that can be defined as $f = 1/T$. Typical sampling frequencies of digital audio are 8, 16, 32, 48, 11.025, 22.05, and 44.1 kHz (Hz represents the number of samples per second). Based on the Nyquist-Shannon sampling theorem, the sampling frequency must be at least 2 times larger than the band limit of the audio signal in order to be able to reconstruct the original analog signal back from its discrete representation. In the next step, each sample in the discrete audio signal is *quantized* with a bit resolution, which makes each sample be represented by a fixed limited number of bits. Common bit resolution is 8-bit or 16-bit per sample. The overall result is a digital representation of the original audio signal, that is referred to as *digital audio signal* or, if it is just considered as a set of bits, for instance for the purpose of storage and compression, as *digital audio data*.

Audio Coding and Compression

The digitization process described above leads to the basic standard of digital audio representation or *coding* named *Pulse Code Modulation* (PCM), which was developed in 1930-1940s. PCM is also the standard digital audio format in computers and Compact Disc (CD). PCM can be integrated into a widely used WAV format, which consists of the digital audio data and a *header* specifying the sampling frequency, bits per sample, and the number of audio channels.

As a basic audio coding format, PCM keeps all samples obtained from the original audio signal and all bits representing the samples. This format is therefore also referred to as *raw* or *uncompressed*. While it preserves all the information contained in the original analog signal, it is also rather expensive to store. For example, a one-hour *stereo* (A Cambridge Dictionary definition of stereo: a way of recording or playing sound so that it is separated into two signals and produces more natural sound) audio signal with 44.1 kHz sampling rate and 16 bits per sample requires 635MB of digital storage space. To save storage in computers and improve the efficiency of audio transmission, processing and management, *compression* theory and algorithms can be applied to decrease the size of a digital audio signal while still keeping the quality of the signal and communicated information at the acceptable level.

Starting with the variants of PCM, such as *Differential Pulse Code Modulation* (DPCM) and *Adaptive Differential Pulse Code Modulation* (ADPCM), a large number of audio compression approaches have been developed [5]. Some most commonly used approaches include MP3/ACC defined in the MPEG-1/2 standard [2,3], Windows Media Audio (WMA) developed by Microsoft, and RealAudio (RA) developed by RealNetworks. These approaches typically lead to a compressed audio signal being about 1/5 to 1/10 of the size of the PCM format.

Audio Content Analysis

A*udio content analysis* aims at extracting descriptors or *metadata* related to audio content and allowing content-based search, retrieval, management and other user actions performed on audio data. The research in the field of audio content analysis has built on the synergy of many scientific disciplines, such as signal processing, pattern recognition, machine learning, information retrieval, and information theory, and has been conducted in three main directions, namely *audio representation*, *audio segmentation*, and *audio classification*.

Audio representation refers to the extraction of audio signal properties, or *features*, that are representative of the audio signal composition (both in temporal and spectral domain) and audio signal behavior

over time. The extracted features then serve as input into audio segmentation and audio classification. Audio segmentation aims at automatically revealing semantically meaningful temporal segments in an audio signal, which can then be grouped together (using e.g., a *clustering* algorithm) to facilitate search and browsing. Finally, an audio classification algorithm classifies a piece of audio signal into a predefined semantic class, and assigns the corresponding label (e.g., "applause," "action," "highlight," "music") to it for the purpose of text-based search and retrieval.

Audio Retrieval

Audio retrieval aims at retrieving sound samples from a large corpus based on their relation to an input query. Here, the query can be of different types and the expected results may vary depending on the application context. For example, in the *content-based retrieval* scenario, a user may use the text term "applause" to search for the audio clips containing the audio effect "applause." Clearly, the results obtained from audio classification can help annotate the corresponding audio samples, audio segments or audio tracks, and thus facilitate this search and retrieval strategy. However, audio retrieval can also be done by using an audio data stream as a query, i.e., by performing *query-by-example* [6]. For instance, one could aim at retrieving a song and all its variants by simply singing or humming its melody line.

In another retrieval scenario, the user may want to retrieve the exact match to the query or some information related to it. This typically falls into the application domain of *audio fingerprinting* [1]. An audio fingerprint is a highly compact feature-based representation of an audio signal enabling extremely fast search for a match between the signal and a large-scale audio database for the purpose of audio signal identification.

Key Applications

Audio technology is widely used in diverse applications areas, such as broadcasting, telephony, mobile communications, entertainment, gaming, hearing aids, and the management of large-scale audio/music collections.

Cross-references

- ▶ Audio Classification
- ▶ Audio Content Analysis
- ▶ Audio Representation
- ▶ Audio Segmentation
- ▶ Multimedia Data
- ▶ Video

Recommended Reading

1. Haitsma J. and Kalker T. A highly robust audio fingerprinting system with an efficient search strategy. J. New Music Res., 32 (2):211–221, 2003.
2. ISO/IEC 11172-3:1993. Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio, 1993.
3. ISO/IEC 13818-3:1998. Information technology - Generic coding of moving pictures and associated audio information – Part 3: Audio, 1998.
4. Pickles J.O. An Introduction to the Physiology of Hearing. Academic Press, London, UK, 1988.
5. Spanias A., Painter T., and Atti V. Audio Signal Processing and Coding. Wiley, NJ, 2007.
6. Wold E., Blum T., and Wheaton J. Content-based classification, search and retrieval of audio. IEEE Multimed., 3(3):27–36, 1996.
7. Yang X., Wang K., and Shamma S.A.Auditory representations of acoustic signals. IEEE Trans. Inform. Theory, 38:824–839, 1992.

Audio Categorization

- ▶ Audio Classification

Audio Characterization

- ▶ Audio Representation

Audio Classification

Lie Lu[1], Alan Hanjalic[2]
[1]Microsoft Research Asia, Beijing, China
[2]Delft University of Technology, Delft, The Netherlands

Synonyms

Audio categorization; Audio indexing; Audio recognition

Definition

Audio classification aims at classifying a piece of audio signal into one of the pre-defined *semantic classes*. It is

typically realized as a combination of a *learning* step to learn a statistical model of each semantic class, and an *inference* step to estimate which semantic class is closest to the given piece of audio signal.

Historical Background

Audio classification associates *semantic labels* with audio signals, and can also be referred to as *audio indexing, audio categorization* or *audio recognition*. As such, audio classification plays an important role in facilitating search and retrieval in large-scale audio collections (databases). Semantic labels are used to represent semantic classes or *semantic concepts*, which can be defined at different abstraction and complexity levels. Typical examples of basic semantic audio classes are *speech*, *music*, *environmental sounds*, and *silence*, which can be detected rather effectively using the methods like [8,9,10,17]. Examples of mid-level semantic concepts are *key audio effects*, like *applause*, *cheer*, *ball-hit*, *whistling*, *car-racing*, *siren*, *gun-shot*, and *explosion*. Finally, the detection of higher-level semantic concepts, such as sport highlights [1,14–16] and action scenes in movies [3,11], is usually performed by analyzing the sequence of detected key audio effects.

Foundations

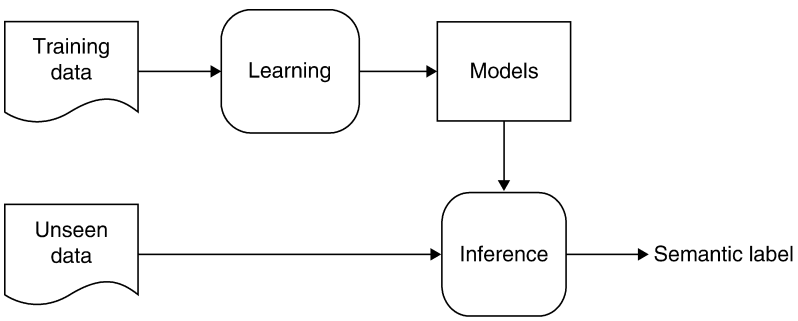
Figure 1 shows a general classification scheme, which is typically composed of two main steps: *learning* and *inference*. In the learning step, a model of each semantic class is built based on a set of training data, and with a specific learning scheme. Then, in the inference step, a new, unseen collection of data is associated with a semantic label, the model of which best resembles the properties of the data. Various schemes have been employed so far for realizing both the learning and inference steps. These schemes include sets of heuristic rules, vector Quantization (VQ), k-nearest neighbor (kNN), decision tree, Gaussian mixture model (GMM), support vector machine (SVM), boosting, Bayesian decision, hidden Markov model (HMM), and neural network. More information about these schemes can be found in [4,5].

While directly applying the previously mentioned learning and inference algorithms works well for straightforward classification tasks, there are a number of issues which should be taken into account when applying this scheme for audio classification. In the following sections, some critical issues are addressed that need to be considered when designing audio classification algorithms, and it is shown on the example of an existing approach how these issues can effectively be resolved. The classification cases discussed in the sections below concern the mid-level semantic concepts (key audio effects) and hierarchies of higher-level semantic concepts.

Key Audio Effect Detection

Several issues play a role in key audio effect detection in a continuous audio signal, and need to be resolved in order to secure reliable classification. The most important issues can be described as follows:

1. Key audio effect detection in a long, continuous audio signal, is typically approached by applying a sliding window of a given length (e.g., 0.5 s) to the signal. The audio segment captured by the window at a given time stamp is then used as the basic unit to associate with a key audio effect. An important implicit assumption here is that each segment corresponds to one and only one semantic class. However, a sliding window is often too short to capture one complete effect, which leads to over-segmentation. The sliding window could also be

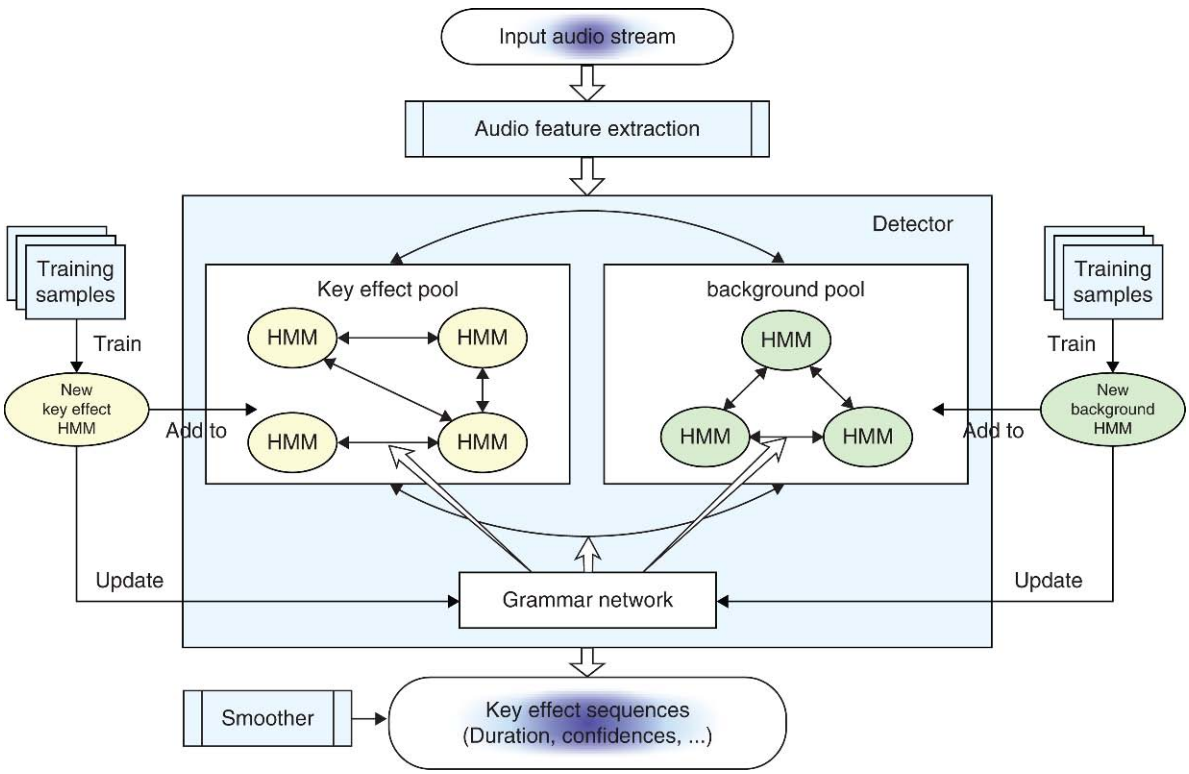


Audio Classification. Figure 1. An illustration of a general classification scheme.

too long and capture several effects within one segment.

2. The targeted audio effects are usually sparsely distributed over the signal, and there are plenty of non-target sounds that are to be rejected. Many existing approaches assume having a complete set of semantic classes available, and classify any audio segment into one of these semantic classes. Other methods use thresholds to discard the sounds with low classification confidence [3]. However, the threshold setting becomes troublesome for a large number of key effects.
3. Audio effects are usually related to each other. For example, some audio effects such as *applause* and *laughter* are likely to happen together, while others are not. Taking into account the transition relationships between audio effects is therefore likely to improve the detection of each individual effect.

To investigate the possibilities for effectively resolving the abovementioned issues when designing algorithms for key audio effect detection, the approach proposed in [2] will be discussed as an example. In this hierarchical, probabilistic framework, as illustrated in Fig. 2, an HMM model is first built for each key audio effect based on the complete set of audio samples, and the defined models are then used to compose the *Key Audio Effect Pool*. Then, comprehensive *background* models are also established to cover all non-target sounds that complement the targeted key effects. Thus, the non-target sounds would be detected as background sounds and excluded from the target audio effect sequence. Moreover, a higher-level probabilistic model is used to connect these individual models with a *Grammar Network*, in which the transition probabilities among various audio effects and background sounds are taken into account for finding the optimal audio effect sequence. Then, for a given input audio stream, the optimal audio effect sequence is found among the candidate paths using the Viterbi algorithm, and the location and duration of each key audio effect in the stream are determined simultaneously, without the need to pre-segment the audio



Audio Classification. Figure 2. The framework for audio effect detection [13], consisting of three main parts: key audio effect pool, background sound pool, and grammar network.

stream into audio segments. In the following both the learning and inference step from [2] are discussed in more detail.

Classifier Learning In the approach from [2], each key audio effect and background sound are modeled using HMMs, since HMM provides a natural and flexible way for modeling time-varying process [12]. The main issue that needs to be resolved for an HMM is the parameter selection, which includes (i) the optimal model size (the number of states); (ii) the number of Gaussian mixtures for each state, and (iii) the topology of the model.

To select the model size for a key audio effect category, one needs to balance the number of hidden states in the HMM and the computational complexity in the learning and inference processes. In general, a sufficient number of states are required to describe all the significant behavioral characteristics of a signal over time. However, when the number of states increases, the computational complexity grows dramatically and more training samples are required. Unlike speech modeling, in which the basic units such as tri-phones could be adopted to specify the number of states, general key audio effects lack such basic units, and thus make the choice of the state numbers difficult. As an example of an approach in this direction, a clustering-based method was proposed in [2,13,17] to estimate a reasonable number of states (model size) per audio effect. The clustering step was realized through an improved, unsupervised *k*-means algorithm, and the resulting number of clusters is taken as the model size.

The number of Gaussian mixtures per state is usually determined experimentally. For instance, the method from [2] adopts 32 Gaussian mixtures for each state in the HMM. This number is larger than those used in other related methods in order to secure a sufficient discriminative ability of the models to identify a large diversity of audio effects in general audio streams.

The most popular HMM topology is the left-to-right or the fully connected one. The left-to-right structure only permits transitions between adjacent states; while the fully connected structure allows transitions between any two states in the model. Different topologies can be used to model audio effects with different properties. For instance, for key audio effects with obvious time-progressive signal behavior, such as *car-crash* and *explosion*, the left-to-right structure should be adopted, while for audio effects without distinct evolution phases, such as *applause* and *cheer*, the fully connected structure is more suitable.

Regarding the background sound modeling, a straightforward approach is to build a large HMM, and train it with as many samples as possible. However, background sounds are very complex and diverse, and their feature vectors are typically widely scattered in the feature space, so that both the number of states and the Gaussian mixtures per state of such a HMM must be particularly large to secure a representation of all possible background sounds. As an alternative, the method from [2] modeled the background sounds as a set of subsets of basic audio classes. It is namely so that in most practical applications the background sounds can be further classified into a few basic categories, such as *speech*, *music*, and other *noise*. Thus, if background models could be trained from all these respective subsets, the training data would be relatively concentrated, and the training time would be reduced. Another advantage of building these subset models is that they could provide additional useful information in high-level semantic inference. For example, *music* is usually used in the background of movies, and *speech* is the most dominant component in talk shows. Following the discussion from above, three background models are built in [2] using the fully connected HMMs for *speech*, *music*, and *noise*. Here, *noise* is referred to as all background sounds except *speech* and *music*. To provide comprehensive descriptions of the background, 10 states and 128 Gaussian mixtures in each state are used in modeling.

The Grammar Network in Figure 2 is an analogy to a language model in speech processing. It organizes all the HMM models for continuous recognition. Two models are connected in the Grammar Network if the corresponding sounds are likely to occur after each other, both within and between the key audio effect pool and the background sound pool. For each connection, the corresponding transition probability is set and taken into account when finding the optimal effect sequence from the input stream.

The transition probabilities between two models can be statistically learned from a set of training data. If no sufficient training data are available, a heuristic approach can be deployed. For instance, the approach presented in [2] is based on the concept of *Audio Effect Groups*, and assumes that (i) only audio effects in the

same group can happen subsequently, (ii) there should be background sounds between any two key audio effects belonging to different groups, and (iii) the transition probability is uniformly distributed per group. An audio effect group can be seen as a set of audio effects that usually occur together. An example Grammar Network with audio effect groups indicated as G_1-G_k is illustrated in Figure 3.

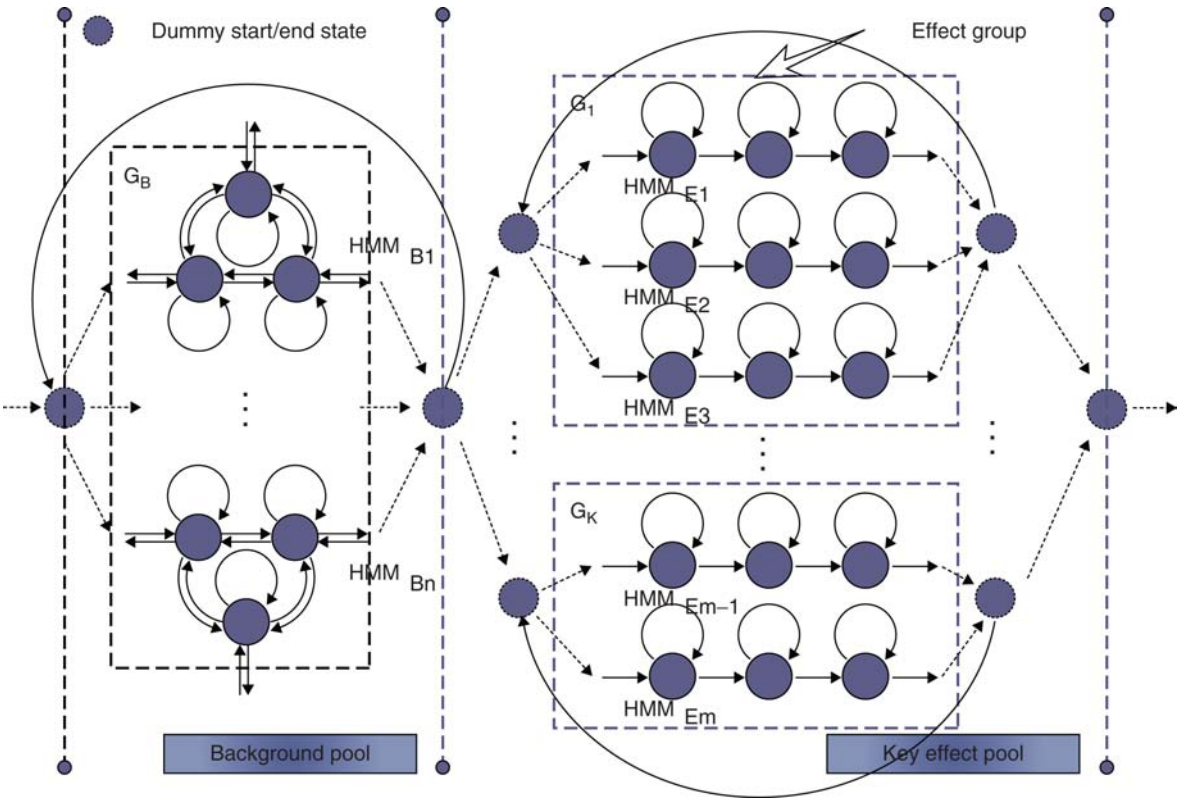
Probabilistic Inference Using the learned classification framework setup described above, the *Viterbi* algorithm can be used to choose the optimal state sequence from the continuous audio stream, as:

$$S_{optimal} = \arg\max_{s} \Pr(s|M, O). \quad (1)$$

Here, s is the candidate state sequence, M represents the hierarchical structure, and O is the observation vector sequence of the input audio stream. In terms of practical realization of this classification scheme, the corresponding state and log-probability are obtained first for each audio frame. Then, a complete audio effect or background sound can be detected by merging adjacent frames belonging to the same sound model. Before this merging step, a smoothing filter can be applied to remove the classification outliers in the sequences of consecutive frames. The final classification confidence can be measured by averaging the log-probabilities of the classified audio frames. In addition, the starting time stamp and duration of each sound occurrence can be obtained simultaneously.

From Key Audio Effects to a Hierarchy of Semantic Concepts

Based on the obtained key audio effect sequence, methods can be developed to perform audio classification at a higher level, such as the level of audio events and scenes. While high-level semantic concepts can generally also be detected using the general scheme from Figure 1, using key audio effects as an



Audio Classification. Figure 3. An illustration of the *Grammar Network* with *Audio Effect Groups*, where G_k is the kth *Effect Group* and G_B is the *Background Sound Pool*. For convenience, all the key audio effect models are presented as 3-state left-to-right HMMs, and all the background models are denoted as 3-state fully connected HMMs. The dummy start and end states are used to link models, and make the structure more clear [2].

intermediate classification level has proved to be a more effective way to perform indexing at this level.

To infer high-level semantics from audio effects, most existing methods are rule-based [1,16], or employ a statistical classification [3,11]. Heuristic inference is straightforward and can be easily applied in practice. However, it is usually laborious to find a proper rule set if the situation is complex. For example, the rules usually involve many thresholds which are difficult to set; some rules may be in conflict with others, and some cases may not be well-covered. People are used to designing rules from a positive view but ignoring those negative instances, thus many false alarms are introduced although high recall can be achieved. Classification-based methods provide solutions from the view of statistical learning. However, the inference performance relies highly on the completeness and the size of the training samples. Without sufficient data, a positive instance not included in the training set will usually be misclassified. Thus these approaches are usually prone to high precision but low recall. Furthermore, it is inconvenient to combine prior knowledge into the classification process in these algorithms.

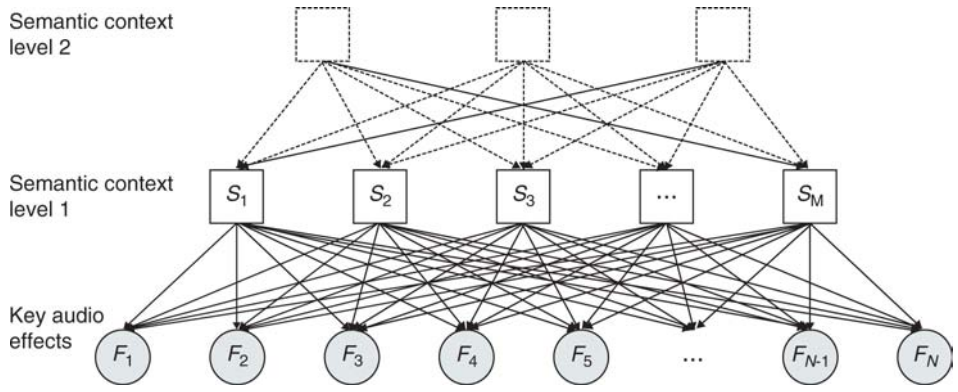
To integrate the advantages of heuristic and statistical learning methods, a Bayesian network-based approach is proposed in [2]. A Bayesian network [6] is a directed acyclic graphical model that encodes probabilistic relationships among nodes which denote random variables related to semantic concepts. A Bayesian network can handle situations where some data entries are missing, as well as avoid the overfitting of training data [6]. Thus, it weakens the influence from unbalanced training samples. Furthermore, a Bayesian network can also integrate prior knowledge by specifying its graphic structure.

Figure 4 illustrates the topology of an example Bayesian network with three layers. Nodes in the bottom layer are the observed audio effects. Nodes in the second layer denote high-level semantic categories such as scenes, while those in the top layer denote much higher semantic concepts. In Figure 4, the nodes in adjacent layers can be fully connected, or partially connected based on the prior knowledge of the application domain. For instance, if it is known *a priori* that some key effects have no relationships with a semantic category, the arcs from that category node to those key effect nodes could be removed. A Bayesian network with a manually specified topology utilizes human knowledge in representing the conditional dependencies among nodes, thus it can describe some cases which are not covered in the training samples.

The nodes in the upper layers are usually assumed to be discrete binaries which represent the presence or absence of a corresponding semantic category, while the nodes in the bottom layer produce continuous values of a Gaussian distribution

$$P(F_i|\mathbf{pa}_i) \sim N(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)(1 \le i \le N) \qquad (2)$$

where F_i is a 2-dimensional observation vector of the ith audio effect and is composed of the normalized duration of an effect and its detection confidence in a given semantic segment. The conditional argument $\mathbf{pa}_i$ denotes a possible assignment of values to the parent nodes of F_i, while $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are the mean and



Audio Classification. Figure 4. An example of a Bayesian network for inference of a hierarchy of semantic concepts. Arcs are drawn from cause to effect. Following the convention, discrete variables are represented as squares while continuous variables are indicated as circles. Furthermore, observed variables are shaded, while hidden variables are not.

covariance of the corresponding Gaussian distribution respectively. In the training phase, all these conditional probability distributions are uniformly initialized and then updated by maximum likelihood estimation using the EM algorithm. In the inference process, the junction tree algorithm [7] can be used to calculate the occurrence probability of each semantic category. Thus, given the information on the audio effects, the semantics in each layer can be inferred based on the posterior probabilities. A semantic segment can be classified into the cth semantic category with the maximum marginal posterior probability:

$$c = \arg\max_{j} \Pr(S_j|F) \quad 1 \le j \le M$$
$$where \quad \mathbf{F} = \{F_1, F_2, \cdots, F_N\} \tag{3}$$

With this scheme, human knowledge and machine learning are effectively combined to perform high-level semantic inference. In other words, the topology of the network can be designed according to the prior knowledge of an application domain, and the optimized model parameters can then be estimated by statistical learning.

Key Applications

Audio classification is typically applied for content-based search and retrieval in and management of large-scale audio collections (databases).

Cross-references

- ► Audio Content Analysis
- ► Audio Representation
- ► Audio Segmentation

Recommended Reading

1. Baillie M., and Jose J.M. Audio-based event detection for sports video. In Proc. 2nd International Conference on Image and Video Retrieval, 2003, pp. 300–309.
2. Cai R., Lu L., Hanjalic A., Zhang H.-J., and Cai L.-H. A flexible framework for key audio effects detection and Auditory context inference. IEEE Trans. Audio Speech Lang. Process., 14 (3):1026–1039, 2006.
3. Cheng W.-H., Chu W.-T., and Wu J.-L. Semantic context detection based on hierarchical audio models. In Proc. 5th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2003, pp. 109–115.
4. Duda R.O., Hart P.E., and Stork D.G., Pattern Classification (2nd edn.), Wiley, New York, 2000.
5. Hastie T., Tibshirani R., and Friedman J. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, New York, NY, 2001.
6. Heckerman D. A tutorial on learning with Bayesian networks. Microsoft Research, Redmond, Washington, Tech. Rep. MSR-TR-95–06, 1995.
7. Huang C. and Darwiche A. Inference in belief networks: a procedural guide. Int. J. Approx. Reason., 15(3):225–263, 1996.
8. Liu Z., Wang Y., and Chen T. Audio feature extraction and analysis for scene segmentation and classification. J. VLSI Signal Process. Syst. Signal Image Video Technol., 20(1–2):61–79, 1998.
9. Lu L., Zhang H.-J., and Jiang H. Content analysis for audio classification and segmentation. IEEE Trans. Speech Audio Process., 10(7):504–516, 2002.
10. Lu L., Zhang H.-J., and Li S. Content-based audio classification and segmentation by using support vector machines. ACM Multimed. Syst. J., 8(6):482–492, 2003.
11. Moncrieff S., Dorai C., and Venkatesh S. Detecting indexical signs in film audio for scene interpretation, In Proc. IEEE Int. Conf. on Multimedia and Expo, 2001, pp. 1192–1195.
12. Rabiner L.R. A tutorial on hidden Markov models and selected applications in speech recognition. Proc. IEEE, 77(2): 257–286, 1989.
13. Reyes-Gomez M.J., and Ellis D.P.W. Selection, parameter estimation, and discriminative training of hidden Markov models for general audio modeling, In Proc. IEEE Int. Conf. on Multimedia and Expo, 2003, pp. 73–76.
14. Rui Y., Gupta A., and Acero A. Automatically extracting highlights for TV baseball programs, Proc. 8th ACM Int. Conf. on Multimedia, 2000, pp. 105–115.
15. Xiong Z., Radhakrishnan R., Divakaran A., and Huang T.S. Audio events detection based highlights extraction from baseball, golf and soccer games in a unified framework, In Proc. IEEE Int. Conf. on Multimedia and Expo, 2003, vol. 3, pp. 401–404.
16. Xu M., Maddage N., Xu C.-S., Kankanhalli M., and Tian Q. Creating audio keywords for event detection in soccer video. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2003, pp. 281–284.
17. Zhang T. and Jay Kuo C.C. Hierarchical system for content-based audio classification and retrieval, In Proc. SPIE: Multimedia Storage and Archiving Systems III, vol. 3527, 1998, pp. 398–409.

Audio Content Analysis

Lie Lu[1], Alan Hanjalic[2]
[1]Microsoft Research Asia, Beijing, China
[2]Delft University of Technology, Delft, The Netherlands

Synonyms

Audio information retrieval; Semantic inference in audio

Definition

An *audio signal* is a signal that contains information in the audible frequency range. Audio content analysis

refers to a set of theories, algorithms and systems that aim at extracting descriptors or metadata related to audio content and allowing search, retrieval and other user actions performed on audio signals.

Historical Background

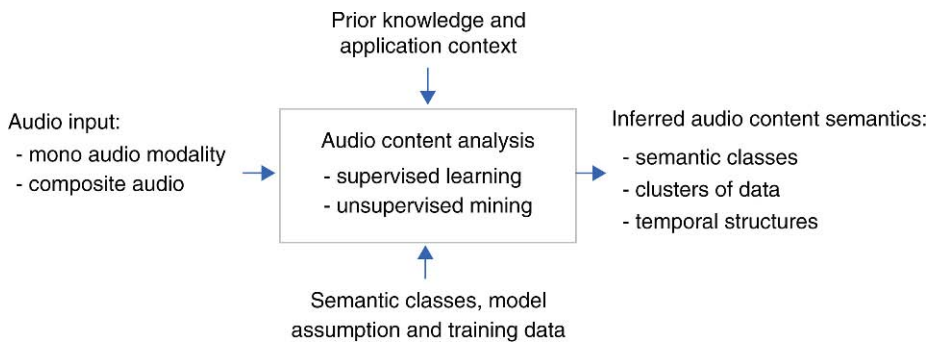
Multimedia content analysis has been one of the most booming research directions in the past years. With the objective of providing fast, natural, intuitive and personalized content-based access to vast multimedia data collections, and building on the synergy of many scientific disciplines, such as signal processing, pattern recognition, machine learning, information retrieval, information theory, natural language processing and psychology, the research initiative born around the end of the 1980s has succeeded in inspiring and mobilizing enormous number of researchers worldwide. This initiative turned into a broad research effort that has continuously gained in strength ever since. As an integrated part of multimedia (multimodal) documents, audio plays an important role in multimedia content analysis. In particular, audio content analysis can be combined with content analysis of visual information to jointly address semantic inference from *multimedia data* streams [5]. However, also taken separately, audio content analysis has been recognized as the key technology for providing easy access to rapidly growing audio archives, and in particular to music collections [3].

Foundations

The underlying problem of audio content analysis is to infer the information about the semantics of the content carried by an audio signal. Here, the audio signal can be a rather simple one consisting of one audio type only (e.g., pure speech, music), or a more complex audio signal (*composite audio*) resulting from a superposition of several audio types, like music, speech, audio effects, and noise. The audio content semantics includes the information on audio types (e.g., speech, music, noise, or any combination of these), on audio semantic classes (e.g., applause, solo instrument, guitar, speaker X or Y), and on audio content structure (e.g., the breaks between semantically coherent audio segments, clusters of *auditory scenes*).

The structure of a general audio content analysis system is illustrated in Figure 1. Depending on the levels at which prior knowledge is specified and the system is trained, this inference system can be realized by employing various approaches ranging from purely supervised to fully unsupervised ones. For example, if the scheme in Figure 1 is seen as a speech recognition system, the prior knowledge, such as the labeled audio data, dictionary and grammar, need to be pre-collected to train both an acoustic model and a language model in a supervised fashion [6]. Similarly, trained models of semantic classes such as car-racing, siren, gun-shot, and explosion can be used to detect the occurrences of these sounds in movie soundtracks [2,4]. Compared to these supervised realizations, an unsupervised approach can be employed to find clusters in audio data corresponding to auditory scenes [1,7], or to find "unusual" events in the sound track of a surveillance signal [9].

While the realizations of the Scheme in Figure 1 can be very different with respect to the types of input audio signals they handle, the prior knowledge and trained models they use, the types of inference techniques they employ (e.g., supervised versus unsupervised), and the inference results they are expected to provide (e.g., semantic categories versus clusters of data), the research targeting these realizations can be



Audio Content Analysis. Figure 1. A general audio content analysis scheme, specifying various possible types of input audio signals, prior knowledge and application context, the types of inference techniques and inference results.

said to follow three main directions that also roughly define the scope of the audio content analysis research field. These directions are *audio representation*, *audio segmentation*, and *audio classification*.

- Audio representation refers to the extraction of audio signal properties, or *features*, that are representative of the audio signal, such as short time energy, zero-crossing rate, pitch, or spectrum. Obtaining a compact feature-based representation of an audio signal can improve the efficiency of audio processing and benefit many applications based on such processing (e.g., audio retrieval).
- Audio segmentation aims to automatically reveal coherent and semantically meaningful temporal segments in an audio signal. Examples of such segments are those containing pure speech or music, or those corresponding to *auditory scenes* [7]. The segments discovered through segmentation can also be clustered together into semantically coherent groups [1] to provide the possibility for an easy content access (e.g., through browsing).
- While the abovementioned segmentation and clustering processes are typically *unsupervised*, audio classification classifies a piece of audio signal into one of the pre-defined semantic classes using *supervised* methods of machine learning and pattern classification. The semantic classes can be defined at the level of basic audio types (e.g., speech, music) [8], audio effects (e.g., applause, gun-shot, car chasing) [2,4], or auditory scenes (e.g., action, highlights, romance).

The term "audio" in the context of audio content analysis typically stands for general or *composite audio* signals [1]. When dealing with specific audio types, such as speech and music, and related applications, dedicated research has been deployed and led to a number of specific research directions like music information retrieval [3] and speech recognition [6].

Key Applications

Audio content analysis is typically applied for content-based search and retrieval in the management of large-scale audio (or multimedia) collections (databases).

Cross-references

- ▶ Audio Classification
- ▶ Audio Content Analysis
- ▶ Audio Representation
- ▶ Audio Segmentation
- ▶ Multimedia Data
- ▶ Video Content Analysis
- ▶ Video Scene and Event Detection

Recommended Reading

1. Cai R., Lu L., and Hanjalic A. Unsupervised Content Discovery in Composite Audio, Proc. IEEE Int. Conf. on Multimedia and Expo, 2005, pp. 628–637.
2. Cai R., Lu L., Hanjalic A, Zhang H.-J., and Cai L.-H. A Flexible Framework for Key Audio Effects Detection and Auditory Context Inference. IEEE Trans. Audio Speech Lang. Process., 14(3):1026–1039, 2006.
3. Casey M., et al. Content-Based Music Information Retrieval: Current Directions and Future Challenges. In Proc. IEEE, Special Issue on Advances in Multimedia Information Retrieval, 96(4): 668–696, 2008.
4. Cheng W.-H., Chu W.-T., and Wu J.-L. Semantic context detection based on hierarchical audio models. In Proc. 5th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2003, pp. 109–115.
5. Hanjalic A. Content-Based Analysis of Digital Video. Kluwer Academic, Norwell, MA, 2004.
6. Huang X, Acero A., and Hon H.W. Spoken Language Processing: A Guide to Theory, Algorithm, and System Development. Prentice, Upper Saddle River, NJ, 2001.
7. Lu L., Cai R., and Hanjalic A. Audio Elements based Auditory Scene Segmentation. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Vol. 5, 2006, pp. 17–20.
8. Lu L., Zhang H.-J., and Jiang H. Content analysis for audio classification and segmentation. IEEE Trans. Speech Audio Process., 10(7):504–516, 2002.
9. Radhakrishnan R., Divakaran A., and Xiong Z. A time series clustering based framework for multimedia mining and summarization using audio features. In Proc. 6th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2004, pp. 157–164.

Audio Feature Extraction

- ▶ Audio Representation

Audio Indexing

- ▶ Audio Classification

Audio Information Retrieval

- ▶ Audio Content Analysis

Audio Metadata

Werner Kriechbaum
IBM Development Lab, Böblingen, Germany

Synonyms

Music metadata

Definition

Audio, first used in 1934 refers to "Sound, esp. recorded or transmitted sound . . . and signals representing this" [Oxford English Dictionary, Oxford 2005, Vol. 1, p, 780].

Metadata is data about data of any sort in any media, describing an individual datum, content item, or a collection of data including multiple content items. In that way metadata facilitates the understanding, characterization, use and management of data.

Audio metadata is structured, encoded data that describes content and representation characteristics of audio entities to facilitate the automatic or semiautomatic identification, discovery, assessment, interpretation, and management of the described entities, as well as their generation, manipulation, and distribution.

Historical Background

Audio metadata predate audio data by centuries. Since antiquity artists and theoreticians alike were interested to classify the effects that could be produced by the combination of different tones and to devise rules for the proper composition of music [12]. The baroque *Affektenlehre* (doctrine of the affections) provided the composers of music with rules and schemes to express affects like love, fear, or hate, and on the other hand gave the listeners a reference system to decode the emotional content of a piece of music. Falling into oblivion with the end of the baroque area similar classification schemes resurfaced with the advent of silent films. Score snippets classified this way enabled the pianist accompanying the film to select music appropriate to the mood of the film scene on the fly.

Like many other technological innovations the modern history of audio metadata started in Bell Labs with the vocoder, developed by Homer Dudley during the late 20s and early 30s of the last century [2]. As part of a speech analysis/synthesis system the vocoder was used to reduce the amount of storage needed to store human speech. The analysis part of the system computed a spectrogram (a variant of a short-term power spectrum) and extracted the fundamental frequency of the speech signal – the first low-level audio metadata. This paved the way for the development of an ever increasing stream of audio analysis techniques that can be traced for example in the "Transactions of the IRE Professional Group on Audio" and its IEEE follow-on and spin-off journals (Available at: http://ieeexplore.ieee.org/xpl/RecentIssue.jsp?punumber = 8340).

MPEG-7 Audio standard [7], finalized in 2002, selected and standardized a set of low- and high-level descriptors from the plethora of available signal processing techniques and tools. MPEG-7 low-level audio descriptors are derived from the time-frequency analysis of the audio signal and include:

- Basic descriptors: e.g., the audio waveform envelope
- Basic spectral descriptors: e.g., the centroid of the audio spectrum
- Signal parameters: e.g., the fundamental frequency
- Timbral temporal descriptors: e.g., the logarithmic attack time
- Timbral spectral descriptors: e.g., the harmonic spectral centroid
- Spectral basis descriptors: which are low-dimensional projections of the spectrum
- Silence

The MPEG-7 high-level descriptors, which are usually collections of further low-level descriptors with additional context, comprise:

- Audio signature
- Musical instrument timbre
- Monophonic melody
- HMM sound model
- Spoken content model

Markup languages and among those most notably HyTime [6], provided the second major force that influenced the development of audio metadata. The linking concepts introduced by HyTime provided mechanisms to connect (textual) symbolic representations and (non-textual) realizations of a media entity without embedding the link in either. The Standard Music Description Language (SMDL) [5] – developed as a HyTime application and up to now not promoted from a draft standard to an international standard – applied these techniques to standardize architecture for the representation of music. From an SMDL point of

view a single musical work is comprised of four domains:

- Logical domain: "The logical domain is the basic musical content – the essence from which all performances and editions of the work are derived, including virtual time values, nominal pitches, etc. The logical domain is describable as 'the composer's intentions with respect to pitches, rhythms, harmonies, dynamics, tempi, articulations, accents, etc.,'" [5, p. 5]
- Gestural domain: Any number of performances of the logical domain, e.g., a digital audio recording capturing a concert
- Visual domain: Any visual rendering of the logical domain, e.g., a score
- Analytical domain: Any number of music-theoretical analyses like e.g., Schenkerian analysis [A. Cadwallader/D. Gagné, Analysis of Tonal Music: A Schenkerian Approach, Oxford 1998]

There is only a faint echo of this concept in the MPEG-7 Audio standard but the ideas are taken up again in the IEEE P1599 Recommended Practice for Definition of a Commonly Acceptable Musical Application using the XML Language (MX) which is as the time of this writing still a draft but should undergo ballot soon. In the terminology of MX, the SMDL domains are called layers and the aspects of music addressed are refined to six:

- General: Information that applies to the piece of music as a whole, e.g., the opus number
- Logic: The symbolic description of the music (equivalent to the SMDL logical domain)
- Structural: Description of music objects and their causal relationship
- Notational: The score (similar to the SMDL visual domain)
- Performance: An audible rendering of the piece of music like e.g., MIDI, CSound, etc
- Audio: Links to and description of digital audio

The great success of the compact disc and the almost ubiquitous availability of tools to create own copies or compilations from digital audio in MP3 format spawned a further community-based audio metadata "standardization" initiative. Since according to the original Red Book specification [IEC 60908 Ed. 2.0 b:1999 Audio recording – Compact disc digital audio system] the compact discs did not include metadata like disc name, track names or author information the need arose to add this information from a supplemental database and to embed at least part of this information in an MP3 file generated from the audio disc. The basic idea to bind the audio disc to its database entry used by all such systems is to compute a hash based on the track information available in the CD's table of contents. This hash is used as key for the metadata stored in the database. An entry in the open-source GPL-licensed audio metadata database freedb [http://www.freedb.org/] contains the following information:

- DISCID: the hash key
- DTITLE: Artist and disc title separated by "/"
- DYEAR: Year the compact disc was released
- DGENRE: The genre in textual form. In addition to this genre entry the database is split into eleven different categories (blues, classical, country, data, folk, jazz, newage, reggae, rock, soundtrack, misc) to minimize hash collisions. The information in the genre field can conflict with the category and will do so when two discs from the same category produce the same hash key. The recommended resolution for the key collision is to put the second disc in another category.
- TTITLEN: The title of track N
- EXTD: Extended data (i.e., any interesting information) for the audio disc
- EXTTN: Extended data for track N

Like the original audio disc the MP3 standard [ISO/IEC 11172–3:1993 Information technology – Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s – Part 3: Audio] did not provide for the inclusion of textual material as part of the encoded audio. This did not deter the user community from enhancing MP3 files with textual information. One result of these efforts is the (informal) ID3 standard [http://www.id3.org/id3v2.4.0-structure and http://www.id3.org/id3v2.4.0-frames] which specifies how to prepend metadata to an MP3 file. There is a rich set of predefined ID3 frames for example:

- Text information frames: Information like album, author, artist(s), etc.
- URL link frames: Links to e.g., the web page of a performer.
- Event timing codes: Time stamps for events in the audio like e.g., verse start, refrain start, theme start, key change, theme end, profanity, profanity end.

- Unsynchronized lyrics or text transcription
- Synchronized lyrics or text.
- Equalization: Equalizer settings for the encoded audio.
- Attached picture

The tag set can be extended since ID3 parsers – like HTML parsers – have to ignore unknown tags.

Foundations

From a metadata point of view, audio is rather ill-defined. All but the basic technical metadata describing the recording setup and the physical metadata characterizing the recorded signal, are specific for the recorded material and require content specific expertise. What makes sense for the description of the recording of a starting airplane is quite different from the data needed to describe birdsong or an opera recording. And even the description of the audio signal on the physical level is not without problems when the metadata are collected for human usage. On their way from the eardrum to the brain, auditory signals undergo a non-linear transformation caused by one's sound perception system [3,9] and all the concepts our cognition forms about audio events are based on this transformed signal. The perceived intensity (loudness) of a signal, for example, is quite different from the physical intensity of the signal. Low-level descriptors of audio signals have to take these differences into account, especially when they are used to derive more complex cognitive descriptors.

Since the domain of the recorded material has a marked influence on the metadata useful and necessary to describe the audio recording, the following discussion is restricted to the recording of music. Almost all music, except monophonic works, is realized by the co-operation of musicians that perform in parallel different subsets of the music. But in many cases, grouping mechanisms in one's auditory perception transform even monophonic music in two or more perceived separate streams [1]. Therefore even pure music not accompanying a stage play can be understood and described as multimedia data and the scientific fundamentals discussed in the entry on multimedia metadata apply to music as well. Furthermore all music can exist in two different forms: a symbolic representation (the score), and a realization (the audio recording). Information like metrum or key that is hard to derive from the analysis of the recorded audio is readily available from the score. In addition a score is accessible for music theoretical analysis that leads to further metadata [8]. Whenever one of the two, score or realization, is missing, it can, at least in principle, be derived from the other. The realizations generated by performing a score vary to a considerable degree. First of all most scores do not give an absolute reference point for the frequency. To map the note A4 (the A above the middle C) to a frequency of 440 Hertz, now an ISO standard [ISO 16:1975 Acoustics – Standard tuning frequency (Standard musical pitch)], is a rather new convention established by an international conference in 1936. Throughout history a variety of reference frequencies for A4 have been used, varying from as low as 392 Hertz up to 466 Hertz. Prior to the acceptance of equal temperament as tuning standard in the second half of the eighteenth century, a variety of tuning standards were in use and rendering baroque music on historic instruments with a historic temperament leads to a quite different performance than the one produced by a modern orchestra with well-tempered tuning. Similar variability exists for the global tempo (at least prior to the use of the metronome ticks to specify absolute time values) and the dynamic. And of course each individual performer or orchestra has its personal style of phrasings and embellishments. Therefore one score gives rise to a variety of realizations and at least when one is interested in identifying music all metadata derived from the realization of the score should in the end lead to the same piece of music. But scores themselves are by no means static; throughout history they have been transformed to adapt them to different needs. Examples of such transformations are transpositions (shifts in pitch) to adapt the score to the ambitus of an instrument, or piano reductions where an orchestral piece is simplified in such a way that its "essence" can be rendered on a piano. Like with the variation in realizations metadata for the description of music should be able to cope with this variability. Both types of variation exist not only in classical western music but in popular music or non-western music like Indonesian gamelan or Indian ragas as well.

Music has a complex temporal organisation and a rich semantic structure that defines a natural segmentation for the audio stream. Like in images, in music many features are characteristic for segments, vary from segment to segment, and become meaningless when averaged over all segments. Musical structure is

not arbitrary but conforms to a set of possible patterns: The sonata form [10] for example is one of the most influential structural patterns during the classical era of western music. The structure of the sonata form is built from three to five pieces: an optional introduction, an exposition, a middle part (*Durchführung*), a repeat, and an optional coda. Usually these five elements are not atomic but further structured and some segments of this substructure are derived from each other by transformations like e.g., transposition, inversion, reflection, or tempo changes. Besides being metadata in its own right, structural information linked with the audio material allows a natural navigation of the recorded performance. As outlined above, the approach to document structure is prescriptive. As in nineteenth century music theory it is assumed that there is an ideal architecture for a sonata form, and that a piece of music not conforming to these rules is in error. In the twentieth century this concept has come under considerable criticism [e.g., 11] and a descriptive approach has been advocated. As a consequence, each musical work is likely to have more than one semantic segmentation, depending on the analysis approaches chosen. To be of any use, both a controlled vocabulary describing the structures and an ontology describing the relationships among them are needed. But this is by no means specific to the structure of music. For many other audio metadata standardized controlled vocabularies and ontologies are still lacking. For example without amendment the Dublin Core [http://www.dublincore.org/] term creator, "An entity primarily responsible for making the resource" [http://purl.org/dc/terms/creator], attached to a piece of music makes it hard if not impossible to recognize the specific role of the creator. Felix Weingartner as creator could refer to his role as conductor (he conducted what is believed to be the first complete recording of Beethoven's symphonies), or his role as composer (symphonies, string quartets, operas) or his role as editor (he edited the complete works of Berlioz).

Key Applications

Audio metadata are essential for any search for audio data. In addition music metadata help to reveal similarities in style or structure between different compositions or different realisations of a piece of music. There are many frameworks for the analysis and manipulation of music, two GPLed packages that allow easy experimentation with different metadata concepts for music are CLAM [http://www.clam.iua.upf.edu/] and RUBATO® [http://www.rubato.org/].

Cross-references

- ▶ Image Metadata
- ▶ Multimedia Metadata
- ▶ Video Metadata

Recommended Reading

1. Deutsch D. (ed.) The Psychology of Music, 2nd edn. Academic, San Diego, CA, 1999.
2. Dudley H.W. The vocoder. Bell Labs Rec., 18:122–126, 1939.
3. Handel S. Listening. MIT, Cambridge, MA, 1989.
4. IEEE P1599/D5.0. Draft recommended practice for definition of a commonly acceptable musical application using the XML Language. NY, 2008.
5. ISO/IEC DIS 10743. Standard music description language (SMDL). July, 1995.
6. ISO/IEC 10744:1997. Information technology – Hypermedia/Time-based structuring language (HyTime). Geneva, 1997.
7. ISO/IEC 15938–4:2002. Information technology – Multimedia content description interface – Part 4: Audio. Geneva, 2002.
8. Mazzola G. The Topos of Music. Birkhäuser, Basel, 2002.
9. Moore B. (ed.) Hearing. Academic, San Diego, CA, 1995.
10. Mauser, S. (ed.) Handbuch der musikalischen Gattungen, Laaber 1993 ff.
11. Rosen C. Sonata Forms, 2nd edn. Norton, NY, 1980.
12. Zaminer F. Geschichte der Musiktheorie, Darmstadt 1984 ff.

Audio Parsing

- ▶ Audio Segmentation

Audio Recognition

- ▶ Audio Classification

Audio Representation

Lie Lu[1], Alan Hanjalic[2]
[1]Microsoft Research Asia, Beijing, China
[2]Delft University of Technology, Delft, The Netherlands

Synonyms

Audio feature extraction; Audio characterization

Definition

An *audio signal* is a signal that contains information in the audible frequency range. Audio representation refers to the extraction of audio signal properties, or *features*, that are representative of the audio signal composition (both in temporal and spectral domain) and audio signal behavior over time. *Feature extraction* is typically combined with *feature selection*, through which the best set of features for the intended operation on the audio signal is defined.

Historical Background

Audio feature extraction typically leads to a strongly reduced audio signal representation. Obtaining such representation can improve the efficiency of audio processing and benefit many applications based on such processing. For example, a compact representation of an audio signal in the form of a *fingerprint* can enable extremely fast search for a match between this signal and a large-scale audio database for the purpose of audio signal identification. Further, if audio features are carefully chosen, they can capture the information from the original data that is relevant to subsequent audio signal analysis and processing steps, while leaving out the redundant and irrelevant (noisy) information parts. This possibility to simultaneously improve the efficiency and robustness of audio signal analysis and processing indicates the importance of the *feature selection* step as the basis step in *audio content analysis*, and in particular in *audio segmentation* and *audio classification*.

Foundations

Audio features can be divided into *temporal* and *spectral* features that capture the temporal and spectral characteristics of an audio signal, respectively. In terms of the way the features are extracted, a division into *frame-level* and *window-level* features can be made. An audio *frame* is the elementary temporal segment of the signal, from which features are extracted. The length of an audio frame typically varies between 10 and 50 ms. Due to its short duration, a frame can be said to contain (close-to) stationary signal behavior. The window-level features are extracted from a longer audio segment, comprising a number of consecutive frames, and are typically marked by applying a sliding window to the signal. While most audio features are extracted at the frame level, window-level features are mainly derived from the frame-level features by investigating their variation along the frames within the window, e.g., the mean and standard deviation of frame-level features. This expansion of the frame-level feature consideration from an individual frame to a series of consecutive frames proved to be useful in many applications, which indicates the importance of window-level features.

Table 1 gives an overview of the typical features proposed in literature to perform various operations on audio, and in particular, the audio segmentation and classification. The features in the table are ranked according to the frequency of their usage in literature. Multiple names indicated per row of the table stand for one and the same feature and/or its variants. Also, the notation ***t***, ***s***, ***fl*** and ***wl*** is used to indicate whether a feature is temporal, spectral, frame-level or a window-level feature. The table is followed by detailed descriptions of a subset of the most prominent frame-level and window-level features. Finally, information is provided about the processes of feature normalization and selection to generate optimal feature sets (vectors) for audio representation.

Zero-Crossing Rate

Zero-crossing rate (*ZCR*) is defined as the relative number of times the audio signal crosses the zero-line within a frame. It can be computed using the following expression:

$$ZCR = \frac{1}{2(N-1)} \sum_{m=1}^{N-1} |\mathrm{sgn}[x(m+1)] - \mathrm{sgn}[x(m)]| \quad (1)$$

Here, sgn[] is a sign function, $x(m)$ is the discrete audio signal, $m = 1...N$, and N is the frame length.

The *ZCR* is a computationally simple measure of the general frequency content of a signal, and as such it is particularly useful in characterizing audio signals in terms of the *voiced* and *unvoiced* sound categories. As speech signals are generally composed of alternating voiced and unvoiced sounds, which is not the case in music signals, the variation in the *ZCR* values is expected to be larger for speech signals than for music signals. Due to its discriminative power in separating speech, music and various audio effects, *ZCR* is often employed in audio content analysis algorithms. An illustration of its practical usage can be found in [7,8,10–12,15].

Short Time Energy

Short Time Energy (*STE*) is the total spectral power of a frame. It can be computed from the audio signal directly, as

Audio Representation. Table 1. An overview of audio features most frequently used in literature for the purpose of audio segmentation and classification

Features	Level	Temporal/spectral
Short time energy, root mean square (RMS), spectrum power, volume, loudness	*fl*	*t, s*
Zero crossing rate (ZCR)	*fl*	*t*
Mel-frequency cepstral coefficient (MFCC)	*fl*	*s*
Spectral centroid, brightness, frequency centroid	*fl*	*s*
Bandwidth	*fl*	*s*
Sub-band energy (distribution), sub-band power, band-energy ratio	*fl*	*s*
Short time fundamental frequency, pitch, harmonic frequency	*fl*	*s*
LPC-derived cepstral coefficients (LPCC)	*fl*	*s*
Linear predictive coding (LPC)	*fl*	*s*
Spectral rolloff	*fl*	*s*
Spectral peak	*fl*	*s*
Spectral moments	*fl*	*s*
Spectral flatness	*fl*	*s*
Harmonicity	*fl*	*s*
Harmonicity prominence	*fl*	*s*
Sub-band partial prominence	*fl*	*s*
Wavelet decomposition	*fl*	*s*
MPEG-7 audio features	*fl*	*s*
Spectrum flux	*wl*	*s*
Percentage of low-energy frames, low short-time energy ratio (LSTER), non-silence ratio	*wl*	*t, s*
High ZCR ratio (HZCRR)	*wl*	*t*
Noise frame ratio, noise or non-voice ratio	*wl*	*t*
4 Hz modulation energy	*wl*	*t, s*
Pulse metric	*wl*	*t*

$$STE = \frac{1}{N}\sum_{m=1}^{N}|x(m)|^2, \tag{2}$$

or from its *Discrete Fourier Transform* (*DFT*) coefficients, as

$$STE = \sum_{k=0}^{K/2}|F(k)|^2. \tag{3}$$

Here, $F(k)$ denotes the *DFT* coefficients, $|F(k)|^2$ is the signal power at the discrete frequency k, and K is the order of *DFT*. In [7,14], this energy is computed using the logarithmic expression, to get a measure in (or similar to) decibels.

Similar to *ZCR*, *STE* is also an effective feature for discriminating between speech and music signals. For example, there are more silence (or unvoiced) frames in speech than in music. As a result, the variation of *STE* in speech is in general much higher than in music. An illustration of the practical usage of this feature can be found in [7,10,12,14,15].

Sub-Band Energy Distribution

To further exploit the energy information based on the *STE* feature defined above, the *sub-band energy distribution* (*SBED*) can be computed. This distribution can be obtained by dividing the frequency spectrum into sub-bands, and by computing for each sub-band j the ratio D_j between the energy contained in that sub-band and the total spectral power of the frame:

$$D_j = \frac{1}{STE}\sum_{L_j}^{H_j}|F(k)|^2 \tag{4}$$

Here, L_j and H_j are the lower and upper bound of sub-band j respectively. The sub-band division can be done

in various ways, such as in octave-scale [7] or in mel-scale [1].

Since the spectrum characteristics are rather different for sounds produced by different sources (e.g., human voice, music, environmental noise) the *SBED* feature has often been used for general audio classification [5,10], and, in particular, for discriminating between different sound effects [1,14].

Brightness and Bandwidth

Brightness and *bandwidth* are related to the first- and second-order statistics of the spectrum, respectively. The brightness is the centroid of the spectrum of a frame, and can be defined as:

$$w_c = \frac{\sum_{k=0}^{K/2} k|F(k)|^2}{\sum_{k=0}^{K/2} |F(k)|^2} \tag{5}$$

Bandwidth is the square root of the power-weighted average of the squared difference between the spectral components and the centroid:

$$B = \sqrt{\frac{\sum_{k=0}^{K/2} (k - w_c)^2 |F(k)|^2}{\sum_{k=0}^{K/2} |F(k)|^2}} \tag{6}$$

Brightness and Bandwidth characterize the shape of the spectrum, and roughly indicate the timbre quality of a sound. From this perspective, brightness and bandwidth can provide useful information for audio classification processes [11,14].

Mel-Frequency Cepstral Coefficient (MFCC)

The set of *Mel-Frequency Cepstral Coefficients* [10] is a cepstral representation of the audio signal obtained based on the mel-scaled spectrum. The log spectral amplitudes are first mapped onto the perceptual, logarithmic *mel-scale*, using a triangular band-pass filter bank. Then, the mel-scaled spectrum is transformed into MFCC using the Discrete Cosine Transform (*DCT*).

$$c_x = \sqrt{\frac{2}{K}} \sum_{k=1}^{K} (\log S_k) \cos[n(k - 0.5)\pi/K] \qquad n = 1, 2, ..., L \tag{7}$$

Here, c_n is the n-th MFCC, K is the number of band-pass filters, S_k is the mel-scaled spectrum after passing the k-th triangular band-pass filter, and L is the order of the cepstrum.

MFCC is commonly used in speech recognition and speaker recognition systems. However, *MFCC* also proved to be useful in discriminating between speech and other sound classes, such as music, which explains its wide usage in the audio analysis and processing literature [3,6,12].

Sub-Band Partial Prominence and Harmonicity Prominence

The *Sub-Band Partial Prominence* (*SBPP*) is used to measure whether there are salient frequency components (i.e., *partials*) in a sub-band. In other words, the *SBPP* estimates the existence of prominent partials in sub-bands [1]. It is computed by accumulating the variation between adjacent frequency bins in each sub-band, that is

$$S_p(i) = \frac{1}{H_i - L_i} \sum_{j=L_i}^{H_i - 1} \left| \hat{F}(k+1) - \hat{F}(k) \right| \tag{8}$$

Here, L_i and H_i are the lower and upper boundaries of the ith sub-band respectively, and the value of $S_p(i)$ indicates the corresponding prominence of salient partial components. The *SBPP* value $S_p(i)$ for sub-bands containing salient components is expected to be large. In order to reduce the impact induced by the energy variation over time, the original *DFT* spectral coefficient vector $\boldsymbol{F}$ is first converted to the decibel scale and constrained to the unit L_2-norm [2] to yield the new spectral coefficient vector used on (8):

$$\hat{\boldsymbol{F}} = \frac{10 \log_{10}(\boldsymbol{F})}{\left\| 10 \log_{10}(\boldsymbol{F}) \right\|} \tag{9}$$

If now the property of an ideally harmonic sound (with one dominant fundamental frequency f_0) is considered, its spectral energy is highly concentrated and precisely located at those predicted harmonic positions which are the multiples of the fundamental frequency f_0. To detect this situation, the following three factors could be measured: (i) the energy ratio between the detected harmonics and the whole spectrum; (ii) the deviation between the detected harmonics and predicted positions; and (iii) the concentration degree of the harmonic energy. Based on the above, the *Harmonicity Prominence* (*HP*) was defined in [14] to estimate the harmonic degree of a sound. The *HP* measure takes into account the above three factors and can be defined as

$$H_p = \frac{\sum_{n=1}^{N} E^{(n)} \left(1 - |B_r^{(n)} - f_n| \Big/ 0.5 f_0\right) \left(1 - B_w^{(n)} \Big/ B\right)}{E} \tag{10}$$

Here, $E^{(n)}$ is the energy of the detected nth harmonic contour in the range of $[f_n - f_0/2, f_n + f_0/2]$ and the denominator E is the total spectral energy. The ratio between $E^{(n)}$ and E stands for the first of the three factors identified above. Further, f_n is the nth predicted harmonic position and is defined as

$$f_n = n f_0 \sqrt{1 + \beta(n^2 - 1)} \tag{11}$$

where β is the *inharmonicity modification factor*, and $B_y^{(n)}$ and $B_w^{(n)}$ are the brightness and bandwidth of the nth harmonic contour, respectively. The brightness $B_y^{(n)}$ is used instead of the detected harmonic peak in order to estimate a more accurate frequency center. The bandwidth $B_y^{(n)}$ describes the concentration degree of the nth harmonic. It is normalized by a constant B, which is defined as the bandwidth of an instance where the energy is uniformly distributed in the search range. Thus, the components $\left(1 - |B_y^{(n)} - f_n| / 0.5 f_0\right)$ and $\left(1 - B_w^{(n)} / B\right)$ in the numerator of (10) represent the second and the third factor defined above.

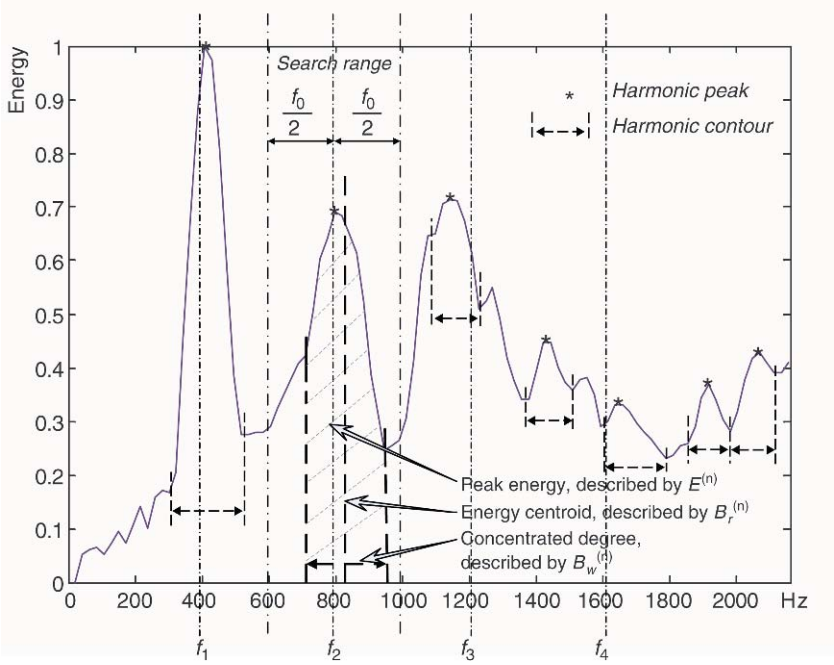
An illustration of the definition of *harmonicity prominence* is shown in Fig. 1. Detailed explanation of the computation and usage of this feature can be found in [1]. The harmonic audio analysis using the *SBPP* and *HP* features enables sophisticated audio classification, like for instance, the discrimination between *cheer* and *laughter*. This is possible because *laughter*, as opposed to *cheer*, usually contains prominent harmonic partials.

High ZCR Ratio

High ZCR Ratio (*HZCRR*) [6] is defined as the fraction of frames in the analysis window, whose *ZCR* values are 50% higher than the average *ZCR* computed in the window, that is

$$HZCRR = \frac{1}{2N} \sum_{n=0}^{N-1} [\mathrm{sgn}(ZCR(n) - 1.5 avZCR) + 1] \tag{12}$$

Here, n is the frame index, $ZCR(n)$ is the zero-crossing rate at the n-th frame, N is the total number of frames,



Audio Representation. Figure 1. Definition of *harmonicity prominence*. The horizontal axis represents the frequency, and the vertical axis denotes the energy. The harmonic contour is the segment between the adjacent valleys separating the harmonic peaks. Based on the harmonic contour, three factors, that is, the peak energy, energy centroid (*brightness*) and degree of concentration (*bandwidth*), are computed to estimate the *harmonicity prominence*, as illustrated at the second harmonic in this example.

avZCR is the average *ZCR* in the analysis window, and sgn[] is a sign function. The variation of *HZCRR* is expected to be higher in speech signals than in music. Fig. 2(I) shows the probability distribution curves of *HZCRR* computed for a large number of speech and music signals. Using the cross-point of two displayed *HZCRR* curves as the threshold to discriminate speech and music leads to the classification error of 19.36%, as shown in [6].

Low Short-Time Energy Ratio

As an analogy for selecting *HZCRR* to model the variations of the ZCR within the analysis window, the *low short-time energy ratio* (*LSTER*) [6,11] can be defined to model the variation of the *short-time energy* (*STE*) in this window. *LSTER* is defined as the fraction of the frames within the analysis window, whose *STE* values are less than a half of the average *STE* measured in the window, that is,

$$LSTER = \frac{1}{2N}\sum_{n=0}^{N-1}[\mathrm{sgn}(0.5avSTE - STE(n)) + 1] \tag{13}$$

Here, N is the total number of frames in the analysis window, $STE(n)$ is the short time energy at the n-th frame, and *avSTE* is the average *STE* in the window. The *LSTER* measure of speech is expected to be much higher than that of music. This can be seen clearly from the probability distribution curves of *LSTER* obtained for a large number of speech and music signals, as illustrated in the Fig. 2(II). Using the cross-point of two displayed *LSTER* curves as the threshold to discriminate speech and music leads to the classification error of 8.27%, as presented in [6].

Spectrum Flux

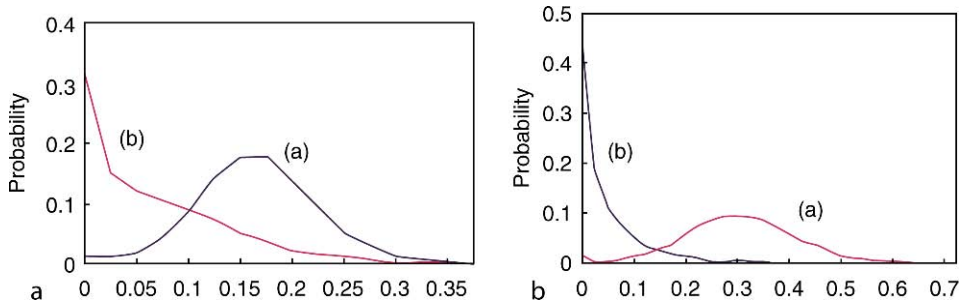
Spectrum Flux (*SF*) is defined as the average variation of the spectrum between adjacent two frames in the analysis window, that is

$$SF = \frac{1}{(N-1)(K-1)}\sum_{n=1}^{N-1}\sum_{k=1}^{K-1}[\log(A(n,k)+\delta) - \log(A(n-1,k)+\delta]^2 \tag{14}$$

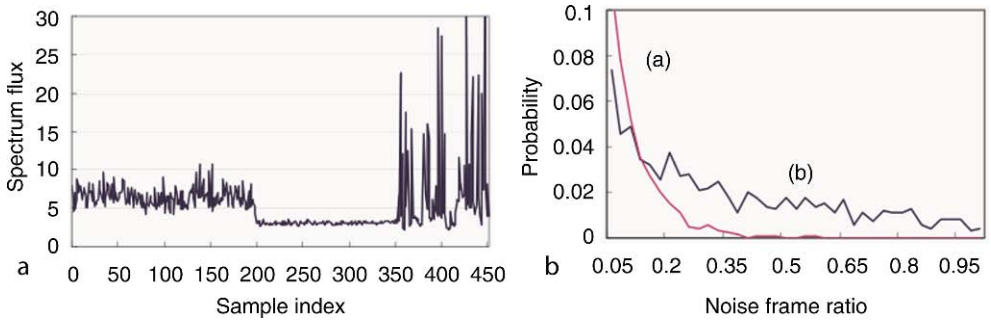
Here, $A(n, k)$ is the absolute value of the k-th *DFT* coefficient of the n-th frame, K is the order of DFT, δ is a very small value used to avoid computation overflow, and N is the number of frames in the analysis window. The *SF* of speech is expected to be larger than that of music. It was also found that the spectrum flux of environmental sounds is generally very high, and that it changes more dynamically than for speech and music [6]. To illustrate this, Fig. 3(I) shows the *SF* computed for an audio segment consisting of speech (0–200 s), music (201–350 s) and environmental sounds (351–450 s). Usage examples of this feature are provided in [8,6,11].

Noise Frame Ratio

Noise frame ratio (*NFR*) is defined as the ratio of noise frames in a given audio clip. A frame is considered as a noise frame if the maximum local peak of its normalized correlation function is lower than a pre-set threshold. The *NFR* is usually used to discriminate environmental sound from music and speech, and to detect noisy sounds. For example, the *NFR* value of a noise-like environmental sound is higher than that of music, because it contains many more noise frames. As can be observed in Fig. 3(II), considering higher NFR values can prove quite discriminative



Audio Representation. Figure 2. An illustration of probability distribution curves of (I) *HZCRR* (a) speech and (b) music, and (II) *LSTER* (a) speech and (b) music.



Audio Representation. Figure 3. (I) The spectrum flux curve of speech (0-200 s), music (201-350 s) and environmental sounds (351-450 s); (II) The probability distribution curves of *NFR*: (a) music and (b) environmental sound.

in separating these two types of audio. An illustration of the usage of this feature can be found in [5,6].

Feature Vector Generation

After they are extracted, features need to be combined together to form a complete audio representation and to provide input into audio analysis and processing steps. Since the values and dynamics of these features may vary considerably over the feature set, simply concatenating them all into a long feature vector is not likely to lead to good results. Therefore, a *normalization* needs to be performed on the features first to equalize their scales. The normalization is usually performed using the mean and standard deviation per feature, namely as $x_i^{'} = (x_i - \mu_i)/\sigma_i$, where x_i is the i-th feature, and where the corresponding mean μ_i and standard deviation σ_i can be obtained from the analyzed data set. Next to the normalization, *feature selection* is usually performed to improve the effectiveness of the feature vector while minimizing its dimension. While feature selection can be realized in many ways [4], a typical approach involves the *principle component analysis* (*PCA*) [13]. Technically, *PCA* is an orthogonal linear transformation that transforms the data to a new coordinate system to reveal the main characteristics (*principal components*) of the data that contribute most to the variance in data, and therefore best explain the data. The principal components can be obtained by performing a covariance analysis or *singular value decomposition* (*SVD*) [13]. If X' is a set of N-dimensional normalized feature vectors from M segments (usually $M >> N$), then X' can be written as an $M \times N$ matrix, where each row corresponds to a feature vector of one audio segment. By applying the *SVD*, the matrix X' can be written as

$$X' = USV^T \tag{15}$$

In terms of *SVD*, V and U are, respectively, an $N \times N$ and $M \times N$ matrix containing the right and left singular vectors, while the diagonal $N \times N$ matrix $S = \text{diag}\{\lambda_1,...,\lambda_N,\}$ contains singular values, with $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_N$. In terms of *PCA*, singular vectors (columns) of the matrix V can be seen as principal components of X', each of which has its corresponding singular value. The larger this singular value is, the more principal (or more important) the component is. Assuming that V_m is a matrix keeping the first m principal components (by keeping the first m columns from V), the original feature set X' can be replaced by a reduced, PCA-transformed feature set

$$X'' = X'V_m \tag{16}$$

which only preserves those features that are relevant to subsequent audio signal analysis and processing steps, while leaving out the redundant and irrelevant (noisy) features.

Key Applications

Audio representation provides fundamentals for audio classification and audio segmentation.

Cross-references

- ▶ Audio Classification
- ▶ Audio Content Analysis
- ▶ Audio Segmentation

Recommended Reading

1. Cai R., Lu L., Hanjalic A., Zhang H.-J., and Cai L.-H. A flexible framework for key audio effects detection and auditory context

inference. IEEE Trans. Audio, Speech Lang. Process., 14 (3):1026–1039, 2006.
2. Casey M.A. MPEG-7 sound-recognition tools. IEEE Trans. Circuits and Syst. for Video Tech., 11(6):737–747, 1997.
3. Foote J. Content-based retrieval of music and audio. In Proc. SPIE Multimedia Storage and Archiving Systems II. 1997, pp. 138–147.
4. Guyon I. and Elisseeff A. An introduction to variable and feature selection. J. Mach. Learn. Res., 3:1157–1182, 2003.
5. Liu Z., Wang Y., and Chen T. Audio feature extraction and analysis for scene segmentation and classification. J. VLSI Signal Process. Sys., 20(1–2):61–79, 1998.
6. Lu L., Zhang H.-J., and Jiang H. Content analysis for audio classification and segmentation. IEEE Trans. Speech Audio Process., 10(7):504–516, 2002.
7. Lu L., Zhang H.-J., and Li S. Content-based audio classification and segmentation by using support vector machines. ACM Multimedia Sys. J., 8(6):482–492, March, 2003.
8. Peltonen V., Tuomi J., Klapuri A.P., Huopaniemi J., and Sorsa T. Computational auditory scene recognition. In Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing, Vol. 2, 2002, pp. 1941–1944.
9. Rabiner L. and Juang B.H. Fundamentals of Speech Recognition, Prentice-Hall, Englewood Cliffs, New Jersey, 1993.
10. Saunders J. Real-time discrimination of broadcast speech/music. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Vol. 2, 1996, pp. 993–996.
11. Scheirer E. and Slaney M. Construction and evaluation of a robust multifeature music/speech discriminator. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, Vol. 2, 1997, pp. 1331–1334.
12. Tzanetakis G. and Cook P. Marsyas: A framework for audio analysis. Organized Sound, 4(3):2000.
13. Wall M.E., Rechtsteiner A., and Rocha L.M. Singular value decomposition and principal component analysis. In A Practical Approach to Microarray Data Analysis, D.P. Berrar, W. Dubitzky, M. Granzow (eds.). Kluwer, Norwell, MA (2003). pp. 91–109, LANL LA-UR-02-4001.
14. Wold E., Blum T. and Wheaton J. Content-based classification, search and retrieval of audio. IEEE Multimedia, 3(3):27–36, 1996.
15. Zhang T. and Kuo C.-C.J. Video content parsing based on combined audio and visual information. In Proc. SPIE: Multimedia Storage and Archiving Systems, IV, 1999, pp. 78–89.

Audio Segmentation

Lie Lu[1], Alan Hanjalic[2]
[1]Microsoft Research Asia, Beijing, China
[2]Delft University of Technology, Delft, The Netherlands

Synonyms

Audio parsing; Auditory scene detection

Definition

Audio segmentation refers to the class of theories and algorithms designed to automatically reveal semantically meaningful temporal segments in an audio signal, also referred to as *auditory scenes* [7]. These scenes can be seen as equivalents of paragraphs in text, and can serve as input into audio categorization processes, either supervised (audio classification) or unsupervised (audio clustering). Through these processes, semantically similar auditory scenes can be grouped together and/or labeled using semantic indexes to provide multi-level, non-linear content-based access to large audio documents and collections.

Historical Background

Automatic detection of *auditory scenes* is an important step in enabling high-level semantic inference from general audio signals, and can benefit various content-based applications involving both audio and multimodal (multimedia) data sets. Traditional approaches to audio segmentation usually rely on a direct analysis of low-level audio features, that is, the targeted audio segments were often defined to coincide with a consistent low-level feature behavior [2,10,11]. This idea served as a basis for numerous approaches for audio segmentation. For example, in [10], a method for scene segmentation was presented that uses low-level features, such as cepstral and cochlear decomposition, combined with the listener model and various time scales. Motivated by the known limitations of traditional low-level feature based approaches, an approach was proposed in [7] to discover auditory scenes based on an analysis of *audio elements*, which can be seen as equivalents to the words in a text document. In this approach that draws an analogy to text document analysis, an audio track is described as a sequence of audio elements, and auditory scenes are segmented based on the semantic affinity among these audio elements and their co-occurrence.

Foundations

Traditional approaches to audio parsing relying directly on audio features have proved effective for many applications, and in particular for those where knowledge on the basic audio modalities (speech, music, and noise) is critical. However, for other applications, like those where higher-level content categories, e.g., semantic concepts, become interesting, the low-level feature based approaches have shown deficiencies due to their incapability of capturing the entire content

diversity of a typical semantic concept. The audio segments obtained by typical feature-based approaches are short and of no higher semantic meaning, if compared to the true semantic segments, like, for instance, *logical story units* targeted by the algorithms of high-level video parsing [3], or the paragraphs in a text document.

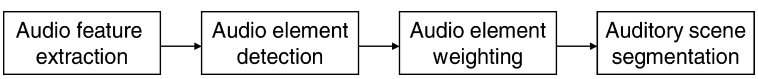
To come closer to the level of auditory scenes, a promising alternative is to design and employ suitable mid-level audio content representations. Figure 1 shows the framework for audio segmentation [6] where the input audio is first decomposed into various *audio elements* such as speech, music, various audio effects and any combination of these. Then, *audio element weighting* is performed to reveal the importance of an audio element to represent an audio document or any of its parts. The audio elements with highest weights can be adopted as the *key audio elements*, being the most characteristic for the semantics of the analyzed audio data [1,4,5,8,9,12–14]. Finally, auditory scenes can be characterized and detected based on the audio elements they contain, just as the paragraphs of a text document can be characterized and detected using a vector of words and their weights. As shown in [7], introducing the mid-level audio content representation in the form of audio elements enables splitting the semantics inference process into two steps, which leads to more robustness compared to inferring the semantics from low-level features directly.

The usefulness of audio elements for audio content analysis was already recognized, e.g., in [13], where the audio elements such as *applause*, *ball-hit*, and *whistling*, are extracted and used to detect the highlights in sports videos. However, this and similar methods usually adopted supervised data analysis and classification methods. There, the scene categories and the corresponding audio elements need to be predefined, which is usually difficult to do for general audio documents. Further, the effectiveness of supervised approaches relies heavily on the quality and quantity of the training data. This makes such approaches difficult to generalize. In view of this, a number of unsupervised approaches were proposed, including the approach to audio element discovery [1], and an approach to auditory scene segmentation [7]. The latter exploits the co-occurrence phenomena among audio elements to realize the segmentation scheme from Fig. 1. This is based on the rationale that, in general, some audio elements will rarely occur together in the same semantic context. On the other hand, the auditory scenes with similar semantics usually contain similar sets of typical audio elements. For example, many action scenes may contain *gunshots* and *explosions*, while a typical scene in a situation comedy may be characterized by a combination of *applause*, *laughter*, *speech*, and *light music*.

Audio Elements Detection and Weighting

As proposed in [1], an iterative *spectral clustering* method can be used to decompose an audio document into audio elements. Spectral clustering can be seen as an optimization problem of grouping similar data based on eigenvectors of a (possibly normalized) affinity matrix. Ng et al. [9] proposed a method to use k eigenvectors simultaneously to partition the data into k clusters, and successfully applied this to a number of complicated clustering problems. To further improve the robustness of the clustering process, the self-tuning strategy [14] can be adopted to set the context-based scaling factors for different data densities. This removes the need for the assumption that each cluster in the input data has a similar distribution density in the feature space, which is inherent in the standard spectral clustering algorithm, but usually not satisfied in complex audio data. Using this clustering method, short audio segments (e.g., one second in length [1]) can be grouped into natural semantic clusters that can then be adopted as audio elements.

In the next step, the obtained audio elements are assigned the importance weights to indicate their prominence in characterizing the content of audio data. Here, two cases can be considered. The first case assumes that only one audio document is available for weight computation. Then, a number of heuristic importance indicators, including *Element Frequency*, *Element Duration*, and *Average Element Length*, are



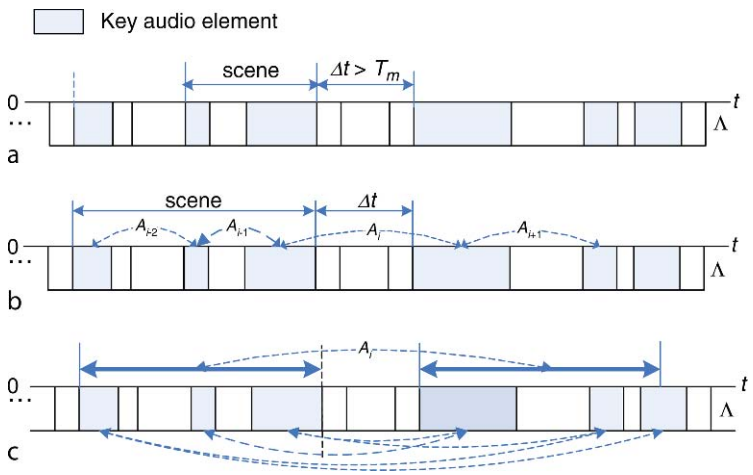
Audio Segmentation. Figure 1. The framework for audio segmentation based on audio elements [6].

proposed in [1] to compute the weight. The second case assumes that multiple audio documents are available to learn the weights. In this case, inspired by the effectiveness of *term frequency* (*TF*) and *inverse document frequency* (*IDF*) used for word weighting in text document analysis, the equivalents of these measures can be defined and employed for the case of audio segmentation. As described in [1,4,5,8,9,12–14], four factors, including *expected term frequency* (*ETF*), *expected inverse document frequency* (*EIDF*), *expected term duration* (*ETD*), and *expected inverse document duration* (*EIDD*), can be defined and combined together to give the importance weight of each audio element. These factors take into account the discriminative power of the occurrence frequency and the duration of a particular audio element to characterize the semantics of an audio document.

Auditory Scene Segmentation

In view of the way it is defined, an auditory scene may consist of multiple, concatenated and semantically related audio elements. An example of such an auditory scene is a *humor* scene consisting of several interleaved segments of *speech*, *laughter*, *cheer*, and possibly also some *light music*. In [6], a simple segmentation scheme was presented that employs crisply defined key audio elements. As shown in Figure 2a, two adjacent key audio elements are assumed to be in the same auditory scene if the time interval between them is sufficiently short. Then the scene boundaries are aligned to the key audio elements, while the background audio elements between two scenes are discarded. Clearly, the algorithm is quite naive and does not fully exploit the relationship between audio elements and auditory scenes. To improve the detection performance, the notion of *semantic affinity* between two contiguous key audio elements was introduced in [1]. This affinity takes into account both the co-occurrence statistics of these key audio elements and the time interval between them, and was employed in [1] to locate the auditory scene boundaries. As shown in Figure 2b, auditory scene boundaries are found between two key audio elements if their semantic affinity is low.

The performance of the segmentation methods discussed above strongly depends on the definition of a key audio element and the reliability of its detection. Crisply defining key audio elements and detecting them in composite audio documents may be rather difficult due to multiple superimposed audio modalities. Therefore, a more reliable solution would be to work with all audio elements instead, and rely on their importance weights. This idea also follows the analogy to the classical text [4] and video scene segmentation approaches [3,5]. As illustrated in Figure 2c, an approach in this direction would decide about the presence of a scene boundary at the observed time stamp



Audio Segmentation. Figure 2. An illustration of previous approaches to auditory scene segmentation, where a vertical line indicates a detected scene boundary: (a) using time interval between key audio elements; (b) using semantic affinity between neighboring key audio elements; and (c) investigating the relationship of (key) audio elements on a large temporal scale.

based on an investigation of the semantic affinity between audio elements taken from a broader range and surrounding this time stamp.

The possibilities for realizing the audio segmentation idea from Figure 2c are now illustrated on the example of the method proposed in [7]. Here, just like in text document analysis, the measure for semantic affinity is not based on the feature-based similarity between two audio segments, but on their joint ability to represent a semantically coherent piece of audio. With this in mind, the definition of semantic affinity in [7] is based on the following intuitive assumptions:

- Affinity between two audio segments is high if the corresponding audio elements usually occur together.
- The larger the time interval between two audio segments, the lower their affinity.
- The higher the importance weights of the corresponding audio elements, the more important is the role these elements will play in the auditory scene segmentation process, and therefore the more significant the computed semantic affinity value will be.

Figure 3 shows an example audio element sequence, where each temporal block belongs to an audio element and where different classes of audio elements are represented by different colors/grayscales. The semantic affinity between the segments s_i and s_j can now be computed as a function consisting of three components, each of which reflects one of the assumptions stated above. The following measure is proposed in [1]:

$$A(S_i, S_j) = Co(e_i, e_j)\, e^{-T(S_i, S_j)/T_m} p_{e_i}\, p_{e_j} \qquad (1)$$

Here, the notation e_i and e_j is used to indicate the audio element identities of the segments s_i and s_j, that is, to describe their content (e.g., speech, music, noise, or any combination of these). P_{ei} and P_{ej} are the importance weights of audio elements e_i and e_j, while $T(s_i,s_j)$ is the time interval between the audio segments s_i and s_j. Further, T_m is a scaling factor which can be set to 16 s, following the discussions on human memory limit [3]. The exponential expression in (1) is inspired by the content coherence computation formula introduced in [12]. Further, $Co(e_i, e_j)$ stands for the co-occurrence between two audio elements, e_i and e_j, in the entire observed audio document, and measures their joint ability to characterize a semantically coherent auditory scene.

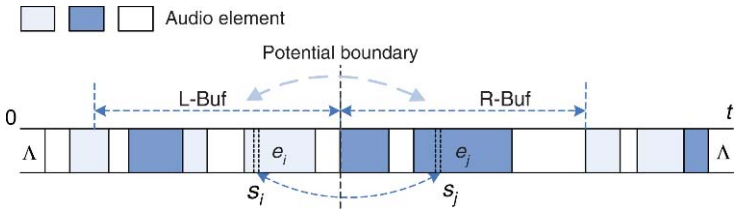
To estimate the co-occurrence between two audio elements, one can rely on the average time interval between two audio elements. The shorter the time interval, the higher the co-occurrence probability is. The procedure for estimating the value $Co(e_i, e_j)$ can then be summarized in the following three steps [1]:

1. First, compute D_{ij}, the average time interval between audio elements e_i and e_j, which is obtained by investigating the co-occurrences of the observed audio elements in the audio signal. For each segment belonging to audio element e_i, the nearest segment corresponding to e_j is located, and then D_{ij} is obtained as the average temporal distance between e_i and e_j.
2. D_{ji} is computed as an analogy to D_{ij}. One should note that D_{ij} is not always equal to D_{ji}.
3. The co-occurrence value can now be found as

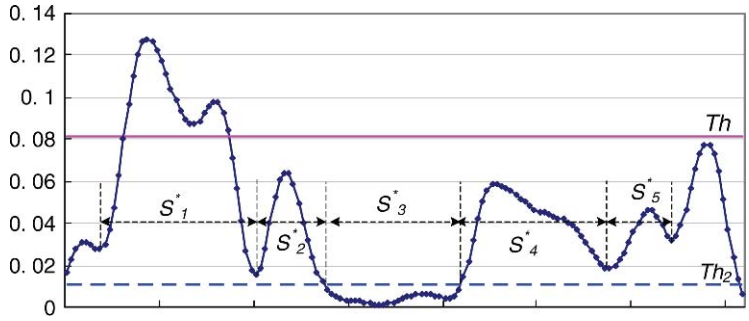
$$Co(e_i, e_j) = e^{-\frac{D_{ij} + D_{ji}}{2\mu_D}} \qquad (2)$$

where μ_D is the average of all D_{ij} and D_{ji} values. The choice for an exponential formula in (2) is made to keep the influence of audio element co-occurrence on the overall semantic affinity comparable with the influence of the time interval between the audio segments (1).

Based on the semantic affinity (1), the confidence of being within an auditory scene at the time stamp t can now be computed by averaging the affinity values



Audio Segmentation. Figure 3. An illustration of an approach to audio segmentation [7], where s_i and s_j are two audio segments, and e_i and e_j are their corresponding audio element identities.



Audio Segmentation. Figure 4. An example of the smoothed confidence curve and the auditory scene segmentation scheme, where S_1^*~S_5^* are five obtained auditory scenes and *Th* and Th_2 are two thresholds.

obtained for all pairs of segments s_i and s_j surrounding the t, that is,

$$C(t) = \frac{1}{N_l N_r} \sum_{i=1}^{N_l} \sum_{j=1}^{N_r} A(S_i, S_j) = \frac{1}{N_l N_r} \sum_{i=1}^{N_l} \sum_{j=1}^{N_r} Co(e_1, e_j) e^{-T(S_i, S_j)/T_m} P_{e_i} P_{e_j} \quad (3)$$

where N_l and N_r are the numbers of audio segments considered left and right from the potential boundary (as captured by the intervals *L-Buf* and *R-Buf* in Figure 3). Using this expression, a confidence curve can be obtained over the timeslots of potential boundaries, as illustrated in Figure 4. The boundaries of auditory scenes can now be detected simply by searching for local minima of the curve. In the approach from [1], the curve is first smoothed by using a median filter and then the auditory scene boundaries are found at places at which the following criteria are fulfilled:

$$C(t) < C(t+1);\ C(t) < C(t-1);\ C(t) < Th \quad (4)$$

Here, the first two conditions secure a local valley, while the last condition prevents high valleys from being detected. The threshold *Th* is set experimentally as $\mu_a + \sigma_a$, where μ_a and σ_a are the mean and standard deviation of the curve, respectively.

The obtained confidence curve is likely to contain long sequences of low confidence values, as shown by the segment S^*_3 in Figure 4. These sequences typically consist of the background audio elements which are weakly related to each other and also have low importance weights. Since it is not reasonable to divide such a sequence into smaller segments, or to merge them into neighboring auditory scenes, one could choose to isolate these sequences by including all consecutive audio segments with low affinity values into a separate auditory scene. Detecting such scenes is an analogy to detecting pauses in speech. Inspired by this, the corresponding threshold (Th_2 in Figure 4) can be set by using an approach similar to background noise level detection in speech analysis [12].

Key Applications

Audio segmentation is typically applied for content-based search and retrieval in and management of large-scale audio collections (databases).

Cross-references

- ▶ Audio Classification
- ▶ Audio Representation
- ▶ Video Content Structure

Recommended Reading

1. Cai R., Lu L., and Hanjalic A. Unsupervised content discovery in composite audio. In Proc. 13th ACM Int. Conf. on Multimedia, 2005, pp. 628–637.
2. Foote J. Automatic audio segmentation using a measure of audio novelty. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2000, pp. 452–455.
3. Hanjalic A., Lagendijk R.L., and Biemond J. Automated high-level movie segmentation for advanced video-retrieval systems. IEEE Trans. Circuits Syst. Video Technol., 9(4):580–588, 1999.
4. Kozima H. Text segmentation based on similarity between words. In Proc. 31st Annual Meeting on Association for Computational Linguistics, 1993, pp. 286–288.
5. Kender J.R. and Yeo B.-L. Video scene segmentation via continuous video coherence. In Proc. IEEE Int. Conf. on Computer Vision and Pattern Recognition, 1998, pp. 367–373.
6. Lu L., Cai R., and Hanjalic A. Towards a unified framework for content-based audio analysis. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 2005, pp. 1069–1072.

7. Lu L., Cai R., and Hanjalic A. Audio elements based auditory scene segmentation. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 2006, pp. 17–20.
8. Lu L. and Hanjalic A. Towards optimal audio keywords detection for audio content analysis and discovery. In Proc. 14th ACM Int. Conf. on Multimedia, 2006, pp. 825–834.
9. Ng A.Y., Jordan M.I., and Weis Y. On spectral clustering: analysis and an algorithm. In Proc. Advances in Neural Information Processing Systems, 2001, pp. 849–856.
10. Sundaram H. and Chang S.-F. Audio scene segmentation using multiple features, models and timescales. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 2000, pp. 2441–2444.
11. Tzanetakis G. and Cook P. Multifeature audio segmentation for browsing and annotation. In Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics, 1999, pp. 103–106.
12. Wang D., Lu L., and Zhang H.-J. Speech segmentation without speech recognition. In Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing, 2003, pp. 468–471.
13. Xu M., Maddage N., Xu C.-S., Kankanhalli M., and Tian Q. Creating audio keywords for event detection in soccer video. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2003, pp. 281–284.
14. Zelnik-Manor L. and Perona P. Self-tuning spectral clustering. In Proc. Advances in Neural Information Processing Systems, 2004, pp. 1601–1608.

Audit Trail

► Logging/Recovery Subsystem

Auditing and Forensic Analysis

Brian Levine, Gerome Miklau
University of Massachusetts, Amherst, MA, USA

Synonyms

Accountability; Monitoring

Definition

The goal of *database auditing* is to retain a secure record of database operations that can be used to verify compliance with desired security policies, to trace policy violations, or to detect anomalous patterns of access. An audit log can contain the authorization ID and time stamp of read and write operations in the database, as well as a record of server connections, login attempts and authorization changes. Government and institutional regulations for the management of sensitive information often require auditing of data disclosure and data modification.

Database forensics is the analysis of the state of a database system to validate hypotheses about past events that are relevant to an alleged crime or violation of policy. Evidence supporting a forensic analysis may be found in an audit log (if available) but may also be recovered from any other component of a database system including table storage, the transaction log, temporary caches, or backup media. A challenge of working with forensically recovered evidence is that it is typically provides only a partial record of an event, possibly based on remnants of deleted information or on inference from incomplete information. On the other hand, it can be difficult to completely eradicate digital evidence from databases, which may be critical for preventing disclosure and complying with policy mandating limited data retention.

While auditing is focused on the preservation and analysis of specific data as required by law or internal corporate policy, digital forensics is broader in scope, potentially relating to any criminal or civil proceeding. Auditors and forensic analysts share some common goals, however the auditor usually relies on information retained intentionally by the system. The forensic analyst is more likely to rely on unintended remnants and inference about past events.

Historical Background

Auditing has been a common practice in settings where sensitive data is managed by computer systems, such as financial and military applications. Auditing has grown in importance as institutions are increasingly required to prove compliance with privacy regulations, or are mandated to discover and publicly respond to exploited vulnerabilities in their systems. Database forensics is an emerging subfield of digital forensics. Computer crime and investigations began receiving attention the late 1970s, but the realization that digital evidence is relevant to a spectrum of crimes has occurred within the last decade. Databases are common components of operating systems, web browers, and email programs and are an important focus of digital investigations.

Foundations

Database Auditing

The auditing component of a database system must support the collection, storage, and protection of

sufficient historical data to enable desired auditing inquiries. Typical auditing queries might include the following:

- Display the query expression for all operations which modified more than ten rows.
- List the authorization IDs of users or client programs who performed SELECT queries on the Patients table between 10 P.M. and 5 A.M. last week.
- List any records in the Employee table whose salary field has been modified more than twice in the past 12 months.

In misuse detection or intrusion detection, the audit log may be used to assess more complex behavior such as: *Is today's workload of update operations similar to "normal" patterns of database usage?*

To support such inquiries, an audit log records the client programs and users who are executing operations, the data objects modified or disclosed, and the context of those operations. For each operation performed, an audit log could contain the SQL query string along with contextual information such as time of day, authorization ID, and network connection information. For update and deletion operations, the audit log may contain the removed values, and the previous values of modified data.

Analyzing database disclosure resulting from a sequence of SELECT queries can be more complex than analyzing the history of database modifications (inserts, updates, and deletes). Auditing disclosure has been the subject of intense research [1,2,4], but faces subtle challenges because a user may be able to infer information not directly released through an executed query. For example, a user's access to an individual database record can be hidden in a sequence of aggregate queries.

In establishing auditing policies, the credibility and completeness of the audit log must be carefully considered to ensure that all relevant events in the system are preserved. For example, the effects of an aborted transaction will be removed from the system for database consistency and atomicity. But a record of aborted transactions, and the reason for abort, could be important to an audit analysis [4].

Naturally, data that will never be relevant to the audit queries under consideration need not be recorded. Further, the retention of recorded audit data must be carefully determined and enforced. The log should be available for appropriate audit inquiries when they arise, but it also contains highly sensitive information and should be destroyed once the period of legitimate inquiry has passed.

Systems Issues and Performance Considerations To support auditing, database systems must efficiently collect required data and permit analysis. Modern commercial databases contain a range of native auditing features which usually include more than one type of log. Common features include a system log to record all connections to the database, and an query log to record each query expression submitted to the database. An auditing policy can be chosen to specify the level of detail that should be logged (e.g., all accesses to relations, or individual tuples; logging of first access in a session or all accesses, etc.).

Logged data may be written to files outside database storage, or to system tables within the database. In the former case, protection depends on operating system access control, while in the latter case protection of audit data depends on the access controls of the database. In particular, the DBA often has privileges to read or alter system tables. When audit data is stored in tables it can have a substantial performance impact on normal database operations.

Database users can implement their own auditing through user-level triggers. A trigger is a user-defined procedure that executes before or after designated events in the database. The triggering event can be an insert, delete, or update command, and most systems allow for tuple-level execution (in which the rule executes once for each tuple affected) or statement level (in which rule executes once for each statement). User-level triggers can be inefficient (especially tuple-level triggers) and the scope of events that can act as trigger events may be limited.

The database transaction log is designed to support critical ACID properties for the concurrent execution of transactions. The transaction log typically includes the before and after images of all database modifications to allow for rollback of aborted transactions and the redo of changes lost due to system failure. While the transaction log contains a wealth of information relevant to auditing, it has a number of limitations when used as an auditing mechanism. First, key data is missing from the transaction log: namely a record of read accesses to the database, as well as some operational context information. In addition, performing an audit analysis using the transaction log could require recovering the state of the database

as of a past moment in time. Although a number of current systems provide such point-in-time recovery by reinstating backups and rolling transactions forward, using this mechanism for audit analysis is very inefficient. Finally, the retention period of transaction log data and audit log data may be substantially different. Transaction logs are often implemented as circular files in which old log records no longer needed for recovery are overwritten. Retention periods for auditing data may be much longer.

In a persistent database (also known as an archiving, or transaction-time database) the historical state of the database is purposely retained as modifications are applied. In such systems a deletion never destroys data and an update merely creates a new version of a tuple. It is possible to pose queries "as-of" any past point in time. Persistent databases have received considerable attention from the research community [5], motivated by both auditing and other applications. Combined with query logs and system logs, a persistent database can offer the most complete audit collection along with efficient audit analysis since the historical state of the database can be queried.

Support for persistence has not been widely implemented, and is not usually used to support auditing in commercial systems. A number of research projects have built persistent, temporal or transaction-time databases, many as extensions to existing systems like MySQL [8], BerkeleyDB [7], and SQL Server.

Protecting the Audit Log It is essential that the audit log be protected from unauthorized modification so that it accurately reflects history. The database administrator, and other privileged parties, should not be capable of tampering with the audit log. Typically there is no legitimate reason for records in an audit log to be modified. The log should be append-only and may be implemented using write-once media. Deletion of the audit log should occur only when it is clear that the log is no longer needed for audit inquiries. Cryptographic techniques have been proposed for detecting tampering of database audit logs and for efficiently tracing the location of illegally modified records once tampering has been discovered.

It is equally critical that the confidentiality of the audit log be protected. The audit log poses multiple privacy threats: the audit log contains records from database that may be sensitive, as well as a history of how the database was used (the users of the database, the queries that were executed, and times of day can all violate the privacy of individuals). Viewing audit logs is a highly privileged operation in most systems. Recent research has investigated the use of cryptography to permit searching over encrypted audit logs to minimize the disclosed data during an investigation [10].

Database Forensics

The goal of database forensics is the analysis of a database system's contents to validate hypotheses about past events that are relevant to an alleged crime or violation of policy. This is a challenge since recovered evidence is typically only a partial record of past events. The goal of the analysis is a formal presentation of recovered data in a court of law, and therefore it is critical for the investigator to also understand legal concepts, including evidence handling. Unlike auditing, there is no limitation on the type of data that can be of interest. Broadly, there are two types of evidence.

- Database evidence can be the direct subject of a crime. For example, records can store contraband, such as images of child pornography, copyrighted media, and stolen intellectual property.
- Evidence can be indirectly related to a crime, for example data from a log that verifies that a relationship exists between two users or computers. Or a log of database query terms can corroborate the notion that a user had knowledge of and intent to possess particular contraband content found in their file system.

Typically investigations cover not only databases but other computer systems (e.g., file systems, email stores, web history) as well as aspects of a crime scene beyond the computer. Evidence from the entire scope of a crime scene must be synthesized and reported as testimony that is persuasive to an adjudicator.

Harvesting Database Evidence Databases are among the most complicated systems found in modern computers. Investigators can harvest evidence from table storage, indexes, transaction and audit logs, temporary caches, database catalogs, or archived copies of records on backed up media.

Data values have a complex lifetime with a database system. When input to a database, records begin in an active state, which means a database and its services need the record in order to function properly. Database operations can change or create active records or remove the purpose of active records. In the latter case, records

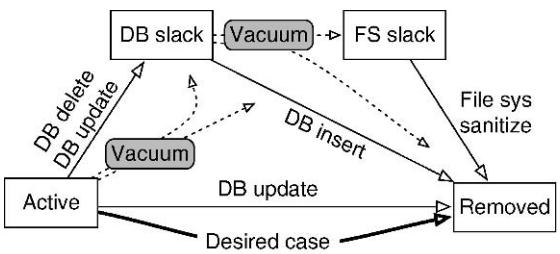
become expired. For example, a record becomes expired when it is deleted and when there is no combination of operations left that would ever use the record again. Forensic investigators are interested in both active and recoverable expired records.

The lifetime of data is illustrated in Fig. 1 for the simple case of records in table storage. An insertion creates an active record, a deletion changes a record from being active to being expired and eventually the data may be overwritten by another database operation. After expiration, a tuple can be either removed, or it can continue to exist as recoverable slack data. Methods of removal are discussed below.

In the case of table storage, each paged file of storage is shared by many records. The database API enables users or investigators to retrieve all active values. When data is deleted by users, typically a single bit is flipped in the page file to indicate removal of the data. However, the record can be retrieved outside the mechanisms of the API.

Other database mechanisms can also leave recoverable records. Updates to records with variable lengths can replace one or more attribute values with smaller attribute values; the tail-end of the old record will remain partially recoverable until it is overwritten. Or an administrator may initiate a *vacuum* command to improve storage performance. When vacuum executes, on many systems, the reorganization is not performed completely in place. In addition to reorganizing records within and across pages, the size of the file used for table storage may be reduced, returning space to the file system, creating the possibility that the database records can be recovered through file system forensics.

Expired data can also be found in stored indexes if, for example, entries in B+tree nodes are deleted but not overwritten immediately. Temporary relations, materialized for improved query processing or used for external sorting, also contain data recoverable as filesystem slack. Data can be recovered from transaction logs, which are typically implemented sequentially written circular files where the newest data overwrites the oldest portion of the log. The amount of recoverable data is dependent on the file system space allocated to the log and characteristics of the database, including the rate and size of updates and checkpoints. Similarly, the amount of data stored in backups varies with policy, storage capacity, and use.



Auditing and Forensic Analysis. Figure 1. The flow of data during its lifetime. It begins in the active state. Before it is deleted and becomes expired it will often be retained as database slack or filesystem slack [9].

System Transparency and Privacy Forensic analysis is not restricted to active tuples because database designs do not strive to eliminate unintended retention of data accessible through interfaces that are not controlled by the database. This incongruence between what the database presents to users and what is actually stored represents a threat to privacy and confidentiality.

For example, as stated above, businesses can unintentionally violate privacy regulations when deleted data is left in table or file storage. Adversaries that investigate databases recovered from lost or stolen computers can reveal sensitive information that was thought to be deleted.

From this point of view, it is desirable for a database to operate in a forensically *transparent way* [9]. Stahlberg et al. have proposed a set of desiderata to determine the extent to which a database system is forensically transparent. A database system is forensically transparent if it satisfies all three desiderata.

Clarity: The impact of each operation on the state of records, whether active or expired, is clear to the user.
Purposeful retention: Only *active* records should be retained by the database.
Complete removal: Expired records must be *removed* by the system within a short, fixed time from when they become expired. In other words, there must be a small upper bound on the time that slack data exists in the database.

Databases that satisfy these desiderata provide a reliable interface to the user in terms of what data is actually stored in the system.

Removal of Data Ensuring that removed data is unrecoverable from a database system is difficult. Records can be removed by overwriting storage with a standard pattern (e.g., all zeros) or through the use of encryption.

Deletion through overwriting can be costly in terms of performance and therefore should be limited

to small records or files. Asynchronous overwriting during idle periods can ameliorate performance degradation while opening a window of opportunity for recovering data. Secure removal of data can also be accomplished by storing data in encrypted form and using overwriting to remove the decryption key. This technique was first proposed for efficient simultaneous removal of data from files and backup logs. Stahlberg et al. [9] present an extended discussion of the use of encryption keys and overwriting in the different internal database mechanisms.

Key Applications

Auditing and forensic analysis are critical operations in any setting where sensitive or high-value data is exposed to untrusted users. For example, database applications that manage financial, national intelligence, or medical information must maintain audit records of past data and operations, and may be the subject of forensic analysis if policy violations occur.

Future Directions

A frequent outcome of auditing and forensic analysis is the detection of a malicious or mistaken operation in the past. Correcting the corrupted database, especially after many valid transactions have been applied, is a significant challenge receiving recent attention by the research community [3,6].

Also note that the above discussion has focused on a single database server. In modern applications, a comprehensive audit or forensic analysis is likely to involve many integrated systems where data objects are derived from diverse sources and follow a complex workflow. In such settings auditing inquiries share some similarities with data provenance (or lineage), which is a record of how a data item has come to exist in a database or view.

Cross-references

► Data Quality (Lineage, Provenance, Curation, Incomplete and Imprecise Data)
► Database Security and Privacy
► Inference Control in Statistical Databases
► Intrusion Detection

Recommended Reading

1. Adam N.R. and Wortmann J.C. Security-control methods for statistical databases: A comparative study. ACM Comput. Surv., 21(4):515–556, 1989.
2. Agrawal R., Bayardo R.J., Faloutsos C., Kiernan J., Rantzau R., and Srikant R. Auditing compliance with a hippocratic database. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 516–527.
3. Ammann P., Jajodia S., and Liu P. Recovery from malicious transactions. IEEE Trans, Knowl. Data Eng., 14(5):1167–1185, 2002.
4. Castano S., Fugini M.G., Martella G., and Samarati P. Database security. ACM/Addison-Wesley, New York, NY, USA, 1994.
5. Jensen C.S., Mark L., and Roussopoulos N. Incremental implementation model for relational databases with transaction time. IEEE Trans. Knowl. Data Eng., 3(4):461–473, 1991.
6. Lomet D., Vagena Z., and Barga R. Recovery from "bad" user transactions. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 337–346.
7. Snodgrass R.T. and Collberg C.S. The τ-BerkeleyDB temporal subsystem. Available at www.cs.arizona.edu/tau/tbdb/.
8. Snodgrass R.T. and Collberg C.S. The τ-MySQL transaction time support. Available at www.cs.arizona.edu/tau/tmysql.
9. Stahlberg P., Miklau G., and Levine B. Threats to privacy in the forensic analysis of database systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 91–102.
10. Waters B., Balfanz D., Durfee G., and Smetters D. Building an encrypted and searchable audit log. In Proc. Network and Dist. Syst. Security Symp., 2004, pp. 91–102.

Auditory Scene Detection

► Audio Segmentation

Authentication

Marina Blanton
University of Notre Dame, Notre Dame, IN, USA

Definition

Authentication is a broad term, which is normally referred to mechanisms of ensuring that entities are who they claim to be or that data has not been manipulated by unauthorized parties. Thus, *entity authentication* or *identification* refers to the means of verifying user identity, after which the user will be granted appropriate privileges. *Data origin authentication* refers to the means of ensuring that the data comes from an authentic source and has not been tampered with during the transmission.

Historical Background

The need for user authentication in early computer systems arose once it became possible to support

multi-user environments. Similarly, data base systems that can be accessed by multiple users with different privileges have to rely on user authentication to enforce proper access control. There is a variety of mechanisms that allow users to authenticate themselves, but password-based authentication is currently the most widely used form of identification.

Data origin authentication (or *data authentication* for short) is also an old concept which gained importance with the adoption of inter-computer communications. With respect to data base systems, data authentication is crucial in distributed environments and when data bases are disseminated to other entities.

Foundations

Identification

The purpose of *entity authentication* or *identification* is to allow one party (the *verifier*) to gather evidence that the identity of another party (the *claimant*) is as claimed. Thus, authentication protocols should permit honest parties to successfully finish the protocol with the claimant's identity being accepted as authentic and make it difficult for dishonest parties to impersonate an identity of another user. Impersonation must remain difficult even for an adversary who can observe a large number of successful executions of the authentication protocol by another entity.

Identification mechanisms can normally be divided into the following types depending on how the identity evidence is gathered:

1. *The user knows a secret.* Such types of identification include passwords, personal identification numbers (PINs), or secret keys.
2. *The user possesses a token.* This is normally based on a hardware token such as magnetic-striped cards (or smartcards) or other custom-designed devices that generate time-variant passwords.
3. *The user has a physical characteristic.* Identification can be based on characteristics inherent to the user being authenticated such as biometrics, handwritten signatures, keystroke dynamics, facial and hand geometries, voice, and others.

Note that for added security often different mechanisms can be combined together. For example, PIN-based authentication is almost always used in conjunction with a physical device that stores information about its owner (i.e., user ID, credit card number, etc.). Likewise, biometric-based recognition can be used in combination with a password or a physical token.

Before a user will be able to engage in an authentication protocol, she needs to *register* with the system and store the data (e.g., a password, keys, biometric data) that will thereafter aid the system in the authentication process. Password-based authentication is treated next, followed by stronger forms of entity authentication.

Password-Based Authentication

Identification based on conventional time-invariant passwords is the most widely used form of identification even though such approaches do not provide strong authentication. A password is a string of (normally 8 or more) characters associated with a certain user, which serves the purpose of a shared secret between the user and the system. When a user initiates the identification process, she supplies the system with the pair (*userid, password*), where *userid* identifies the user and *password* provides the necessary evidence that the user possesses the secret.

The most straightforward approach for the system to store passwords is in the clear text. This, however, allows the system administrator, or an adversary in case of system compromise, to recover passwords of individual users leading to security concerns. Thus, most systems first apply a one-way *hash function* to each password and store the output in the system. Then when a user supplies the password during the identification process, the password is first hashed and then compared to the string stored in the system. This security measure no longer allows cleartext passwords to be recovered, but other attacks on passwords are still possible. In particular, the following attacks can be carried out:

– *Replay of passwords.* Since passwords are reusable, an adversary who obtains password information (by either seeing the user type the password, using a keylogger program, or capturing the password in transit from the user to the system) will be able to reuse it and successfully impersonate the user.
– *Exhaustive search.* An adversary might attempt to guess user passwords by trying all possible strings as potential passwords (on the verifier itself or by obtaining a copy of the password file and performing this attack off-line). Generally it is infeasible for an adversary to try all passwords if they are chosen from a sufficiently large space, but it is

possible to exhaust short passwords (e.g., of length 6 characters or less).
– *Dictionary attack.* It is well known that users tend to choose passwords that they can easily remember but which are considered weak from the security point of view. Thus, an adversary might try to guess a user password using words from a dictionary and variations thereof. If a user makes a poor password choice, such an attack can have a high probability of success. Dictionary attacks become increasingly complicated, testing for combinations of words, common substitutions and misspellings, insertion of additional symbols, words from foreign languages, etc.

To decrease the vulnerability of the system to these attacks, additional measures are normally employed, some of which are:

– *Salting passwords.* In order to make guessing attacks less effective, many systems use an additional random string, called *salt*, with each password. Before a password is stored, it is augmented with a random salt, hashed, and then stored in the system along with the salt. This prevents an attacker who is in possession of a file with many user passwords to launch a dictionary attack against all of them at the same time, and requires each user's password to be tested individually.
– *Slowing down password verification.* To defeat against attacks that perform trials of a large number of passwords, the function that computes the hash of the password can be made more computationally extensive. This, for example, can be done by iterating the computation n times. When increasing the computation for password mapping, a care, however, must be taken not to impose a burden on legitimate users.
– *Limiting the number of unsuccessful password guesses.* It is common for a user account to be locked after the number of unsuccessful authentication attempts exceeds a certain threshold. The owner of a locked account must then contact an administrator and have the account activated.
– *Password rules.* To protect against guessing attacks, often certain rules are imposed on user choice of passwords such as the minimal password length and/or usage of capital letters, numbers, and special symbols. Such rules normally strengthen the password choices but they also limit the password search space. Also, a technique called *password aging* is often employed to force users to choose a new password after a certain period of time. If such a period is rather short, however, users will be unable to remember a newly chosen strong password, thus weakening the security of the system with bad password choices.

It is always a challenge to find a balance between memorability of passwords (passwords that are hard to remember tend to be written down) and their resistance to dictionary attacks (i.e., passwords with enough randomness in them). Thus to aid users in choosing less predictable passwords which they can remember, techniques exist to create computer-generated *pronounceable passwords* which are based on mnemonics. Also, recently various solutions have been developed to use images for authentication (a user is given a number of images and is asked to identify the set of pre-selected images), graphical interfaces (a user draws a pattern on a grid that has to match a previously chosen pattern), etc. Such systems, however, are not widely deployed and have not been thoroughly evaluated to determine the level of security they provide.

Since a major security concern with fixed passwords is the possibility of replaying them, a natural way to improve their security is to consider *one-time passwords.* As the name suggests, in such systems each password is used only once and there are different ways to realize them, which is briefly outlined next.

- The user and the system initially agree on a sequence of secret passwords. Each time the user authenticates to the system, a new password is used. This solution is simple but requires maintenance of the shared list.
- The user updates her password with each instance of the authentication protocol. For instance, the user might send the new password encrypted under a key derived from her current password. This method crucially relies on the correct communication of the new password to the system.
- The new password is derived with each instance of the authentication protocol using a one-way *hash function.* As an example, consider the scheme called S/Key due to Lamport [2]. The user begins with a secret k and applies a one-way *hash function* h to produce a sequence of values $k, h(k), h(h(k)),..., h^t(k)$. The password for ith identification session

is $k_i = h^{t-i}(k)$. When the user authenticates $(i+1)$ st time with k_{i+1}, the server (which has k_i for that user stored) checks whether $h(k_{i+1}) = k_i$ and, if so, accepts the authentication and replaces k_i with k_{i+1}. This check convinces the server because the function h is considered to be infeasible to invert and only the legitimate user will be able to construct k_{i+1} that passes the check.

Challenge-Response Identification

Challenge-response techniques provide a strong form of entity authentication as they are not vulnerable to replay attacks. The main idea behind such protocols is that the claimant possesses a secret. During an identification session, the server sends to the claimant a unique randomly chosen *challenge*. The claimant computes a *response* which is a function of her secret and the server's challenge and sends it to the server. It is important to note that the response does not provide information about the user secret, and cannot be used to successfully compute responses to server's challenges in the future by someone who monitors the message exchange.

A variety of cryptographic challenge-response techniques exist which could be based on (i) symmetric encryption, (ii) one-way *hash functions*, or (iii) public-key encryption. A more detailed explanation of such techniques is beyond the scope of this article and can be found in standard textbooks on cryptography such as [3,7].

Data Origin Authentication

The purpose of *data origin authentication* is to ensure that the data comes from a trusted source and has not been tampered with during the transmission. Techniques that permit verifying data authenticity can be divided in two categories: (i) the communicating parties *share a common secret* and (ii) the communicating parties *do not share a secret*.

Consider that in a distributed database environment two systems communicate often and there is a need to ensure data integrity. Then such systems can share a secret S that permits them to use *message authentication codes* (MAC) for data authentication. That is, prior to transmitting the data, the sender constructs the MAC using S and sends it along with the data. After obtaining the data, the receiver uses the shared secret to reconstruct the MAC and compare it with the MAC received. If the check succeeds, the data is accepted as authentic, and it is discarded otherwise.

In cases when the sender and the receiver do not already have a secret which is known to both of them, *digital signatures* can be used to verify the authenticity of the sender and integrity of the data. This mechanism assumes that the sender has a public-private key pair, which is used to sign the data. Then the sender uses her private key to sign the message and the receiver uses the corresponding public key to verify the fact that the message arrived intact.

The standard way of producing a digital signature on data is first to apply a one-way *hash function* on the data to compute its digest and then sign the digest. The purpose of computing the digest first is to compress the data to a short string, which then can be efficiently signed to produce a fixed-size signature. In cases when integrity of a database needs to be verified with some regularity while only parts of it change, more advanced techniques can be used. In particular, Merkle hash tree is commonly used to produce a digital signature on a hierarchically structured set of documents (e.g., an XML tree of documents). In such a tree, digests of individual nodes are computed and then combined in a bottom-up fashion to result in a single short digest of the tree. The owner of the data produces a signature on the root node only. Verification of data integrity in such trees can normally be done faster than recomputing digests of the entire tree if the user would like to verify the integrity of only a part of the tree.

Key Applications

The main application of authentication is access control. Namely, in multi-user systems a user authenticates to the system and is granted access to specific resources determined by her access privileges. The mechanisms used to determine user access rights vary drastically from one system to another and are based on the type of access control (e.g., role-based, discretionary, etc.,) and access control policies.

Also, authentication applications and services such as Kerberos, X.509 Authentication Service, or Public-Key Infrastructure (PKI) can be used to aid in the authentication process.

In case of data authentication, verification of data integrity and authenticity is essential in determining trustworthiness of the data. For example, updated database records that are coming from a trusted source can be safely used to modify the current contents of the database. If, on the other hand, data integrity and

authenticity of the modifications cannot be verified, in many cases such data will not be trusted.

Cross-references

- ▶ Access Control
- ▶ Digital Signatures
- ▶ Hash Functions
- ▶ Merkle Hash Trees
- ▶ Message Authentication Codes
- ▶ Security Services
- ▶ Storage Security

Recommended Reading

1. Bishop M. Computer Security: Art and Science. Addison Wesley Professional, 2002.
2. Haller N. The S/Key one-time password system. In Proc. Symp. on Network and Distributed System Security, 1994, pp. 151–157.
3. Menezes A., van Oorschot P., and Vanstone S. Handbook of Applied Cryptography. CRC, 1996.
4. Pfleeger C. and Pfleeger S. Security in Computing, 3rd edn. Prentice-Hall, 2003.
5. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd edn. Wiley, 1996.
6. Stallings W. Cryptography and Network Security: Principles and Practices, 4th edn. Pearson Prentice Hall, 2006.
7. Stinson D. Cryptography: Theory and Practice, 3rd edn. Chapman & Hall/CRC, 2006.

Authentication Trees

▶ Merkle Trees

Authorization Administration Policies

▶ Access Control Administration Policies

Authorization Administration Privileges

▶ Access Control Administration Policies

Authorization Policy Languages

▶ Access Control Policy Languages

Authorization Verification

▶ Access Control

Auto-administration and Auto-Tuning of Database Systems

▶ Self-Management Technology in Databases

Auto-Annotation

▶ Automatic Image Annotation

Automata Induction

▶ Grammar Inference

Automatic Abstracting

▶ Summarization

Automatic Image Annotation

Nicolas Hervé, Nozha Boujemaa
INRIA Paris-Rocquencourt, Le Chesnay Cedex, France

Synonyms

Multimedia content enrichment; Image classification; Object detection and recognition;Auto-annotation

Definition

The widespread search engines, in the professional as well as the personal context, used to work on the basis of textual information associated or extracted from indexed documents. Nowadays, most of the exchanged or stored documents have multimedia content. To reduce the technological gap so that these engines still can work on multimedia content, it is very convenient developing methods capable to generate automatically textual annotations and metadata. These methods will then allow to enrich the upcoming new

content or to post-annotate the existing content with additional information extracted automatically if ever this existing content is partly or not annotated.

A broad diversity in the typology of manual annotation is usually found in image databases. Part of them is representing contextual information. The author, date, place or technical shooting conditions are quite frequent. Some semantic or subjective annotations, like emotions that flow out from images, can be found. Some other annotations could be related to the visual content of images. They provide information on a given image such as indicating whether it is a drawing, a map or a photograph. . . For photographs, the global aspect is often specified (vertical/horizontal, color/black and white, indoor/outdoor, landscape, portrait. . .), as well as the presence of remarkable objects or persons.

The aim of automatic image annotation approaches is to provide efficient methods that extract automatically the visual content of pictures allowing semantic labeling of images. This is generally achieved by learning algorithms that, once being trained on annotated sub-corpora, are able to suggest keywords to the archivist through object detection/recognition and image classification methods.

Historical Background

The exploration of visual content databases and their querying to retrieve some specific content usually rely on textual annotations that have been previously provided manually by human operators. The outcome of the tremendous improvements in digitization and acquisition devices is the availability of exponentially growing content. Usual annotation techniques then became more and more difficult to apply because they are time and cost consuming. Moreover, manual annotations are far from being perfect. They are often focused on the context, subjective, partial and driven by the needs of the end-users at the time they are produced. As these needs are evolving, part of the existing annotations becomes irrelevant and others are missing. This is especially true with the arising of Internet and the availability of all kind of databases online. An other issue lies in the lack of controlled vocabularies for most of the databases making difficult for the end-user to guess what query words he has to use in order to retrieve the content he has in mind.

Visual content indexing and retrieval community have achieved significant progress in the recent years [13] toward efficient approaches for visual features extraction and visual appearance modeling together with developing advanced mechanism for interactive visual information retrieval. One of the major issues was and remains the semantic gap [4,8].

Two main types of images databases could be distinguished. Specific databases are focused on a given restricted field. In the scientific domain, one can cite satellite images for weather forecast or cultivation study, medical images or botanical databases for species recognition. They are also found in the cultural heritage domain (e.g., paintings databases) or the military and security domain (e.g., fingerprints and faces databases). On the other side, generic databases contain very different images, without any *a priori* on their content. This is usually the case for professional news agencies, illustration photo stock collections and personal family and holiday photo albums. Only methods for generic content databases labeling will be addressed.

By analyzing automatically images and characterizing them with low-level features (mainly colors, textures and shapes) CBIR systems [3] provided new query paradigms that enable users to express their needs. The main one is "query by example" where the system retrieves images of the database that are the most similar to a given example. The scientific community has been facing the well known semantic gap problem for a while which remain the major concern of the research community. Since the late 90s, relevance feedback mechanism is one of possible solutions to this difficult problem.

The early papers on automatic annotation that have been published tackled image orientation detection or the classical indoor versus outdoor and city versus landscape classifications of photographs [14,15]. Recently, relevance feedback allows moreover helping for interactive mass-annotation of image collections. This approach is often referred to as semi-automatic image annotation.

Foundations

Despite some of its drawbacks, the query by keyword is still very useful and quite natural for the end-user [6]. Automatic annotation generates such keywords to enrich the images semantic descriptions and ease further querying. Because of the computational costs of all current approaches, the existing systems are always composed of two parts. An offline part is in charge of

indexing the visual content and generating the annotations. Eventually, a human operator can help the system during the process or after it to validate/invalidate the produced annotations. In such cases, one talks of semi-automatic annotation systems. The second part, online and real-time, is a query by keywords module.

As the main purpose is to describe the visual content, the term "visual concept" is preferred over keyword to describe the labels a system has to discover in images. As previously mentioned, these visual concepts could be related to either global appearance of the image or presence of some objects. Objects detection can also be refined in generic object class detection or specific object instance detection. For example, one can ask a system to label only the "vehicle" concept, or more precisely to distinguish cars, motorbikes, boats and airplanes, and, at a very specific level, being able to recognize different makes of cars. This is the same problem with annotating persons. Being able to detect the presence of a person in an image is a different procedure and result than recognizing him. As face recognition is a well studied problem which is tackled by a specific research community. The ability to generalize from a few examples and to reach higher abstraction levels is natural for humans but it is very challenging task to achieve with current state-of-the-art's annotation systems [5,7,10,12].

One of the fundamental hypotheses of automatic annotation is that what looks similar is probably semantically similar. Most of the approaches rely on this assumption. The main generic steps of automatic annotation are described below. First, visual features are extracted automatically from images in order to obtain representations in a visual space. The second step is to build models that will link the visual concepts to the relevant information in the visual space. When new content is proposed, models are then able to predict the corresponding visual concepts.

The performances evaluation of such methods may rely on the usage of the annotations by the final users. As in most of information retrieval systems, precision and recall measures are used. Precision emphasizes the retrieval of relevant documents earlier and recall focuses on the retrieval of the full set of relevant documents. Precision and recall are complementary to judge the quality of a system. But for some applications, precision is the only important measure. This is especially the case when a huge image database is available (like Internet). When doing a query, a user is more interested in the first satisfying results than in the complete relevant result set.

Images Description with Low-Level Features

The visual description of images is of great importance as it is the raw material on which further models are built. There is not a universally good low-level features extractor. In specific databases, *a priori* on the content of images can be used to extract specialized features that will better describe their special nature. For example, numerous features can be found in the literature for faces or fingerprints description. In generic databases, compromises have to be made between exhaustiveness, fidelity to the content, ability to generalize and different invariance degrees (illumination changes, rotations, scales, occlusions. . .). The use of inappropriate features leading to poor performances of a system has often been described as semantic gap. In this case, one rather faces the numerical gap, meaning that the visual information is present in images but it has not been extracted correctly.

Due to their ability to generalize to content in different conditions, statistical features are often used. They gather color, shape and texture information in histograms, separately or jointly. Color histograms are among the first features used to describe images. They vary depending on the underlying color space that is used, the quantization parameters, different weighting schemes or the use of co-occurrences of colors. Shapes can be described by properties of edges found in images like their types, orientations or lengths. Textures are focusing on the analysis of frequencies in images. They often rely on Fourier transform, Gabor filter banks or wavelets. Some features also combine different types of information, mixing for example color and texture in a single representation. Typically, these visual features are represented by vectors in high-dimensional spaces (generally between a few tens and a few hundreds dimensions) [9,15].

Initially, the features were extracted over the full image. This approach is well suited to describe the global aspect of the content but is too coarse to represent small details and objects. Features need to be extracted locally. First, a support region has to be determined. Once its location, shape and size are known, features are computed on this small portion of the image. These features can be of the same types as those extracted at a global level or they can be specialized according to the nature of the support regions. Several strategies are used to select the support regions. Segmentation algorithms try to find the boundaries between homogeneous regions in images [2]. Segmentation is a difficult problem in itself that is not well defined. Unfortunately, the general trend

has always been to focus on segmentation that detects objects, which is already a highly semantic task and, thus, not really achievable through automatic processes. Alternative approaches consist on sliding windows and fixed grid, with varying sizes and spacing, are common ways of obtaining dense sampling of the visual content [9,14]. Another popular region selection approach is based on local features detectors via point-of-interest. They were originally designed for image registration. These detectors are generally attracted to specific areas of images that have high variation in the visual signal, such as the vicinity of edges and corners of regions. They allow the selection of a very small proportion of image locations having the highest visual variance [11,16]. Typically, when using dense sampling, point-of-interest or when mixing them, between a few hundreds and a few thousands features are extracted per image. The computational cost is then much higher than with global features. Some representations also try to carry other information, like geometrical relations between features locations or contextual information [1].

With global features, the image representation is straightforward. However, even when local features are to be used, learning algorithms may sometimes require a global image representation that encompasses all the local visual information. The bag-of-visual-words representation, very much inspired by the classical bag-of-words representation for text, is one of the most popular for images. A visual vocabulary composed of visual words (some representative features) is generated. An image is then represented by a coordinate vector, each value of which expresses the degree of importance of a feature with respect to the image and/or the database as a whole. The creation of a visual vocabulary is an important step in the full process. The selected visual words (a few hundreds to a few thousands) have to be representative of the database content as they will serve as a basis for further representation. Creating a good vocabulary will avoid the loss of too much local information. Generally, clustering algorithms either supervised or not, are used with a sample of the database. This step can be seen as a quantization of the local features.

Whatever the selected representation, the similarity between images is measured by a distance functional in the visual space. A broad variety has been developed: classical Euclidean distance (L2), L1, earth mover distance (EMD), chi-squared (χ^2), vector angle, histogram intersection,...

Learning and Models

Although several formulations have been proposed, the main purpose of building models for visual concepts is to associate them with the visual space regions that best represent them. This problem is at the crossroads of computer vision, data mining and machine learning. Usually, the models are built through a supervised learning process. For given visual concepts, an algorithm is fed with a training dataset containing both positive and negative images regarding the concepts to learn. This algorithm has to find the discriminant information from the visual space that best models the concepts. Generally, the available annotations for the training set are not localized. The presence of a visual concept for an image is known, but its exact location is not provided. This is the case for almost all professional and personal databases. In the same way, annotating new images does not require to locate exactly the visual concept, but only to predict its presence. This is the main distinction that can be made with object detection tasks.

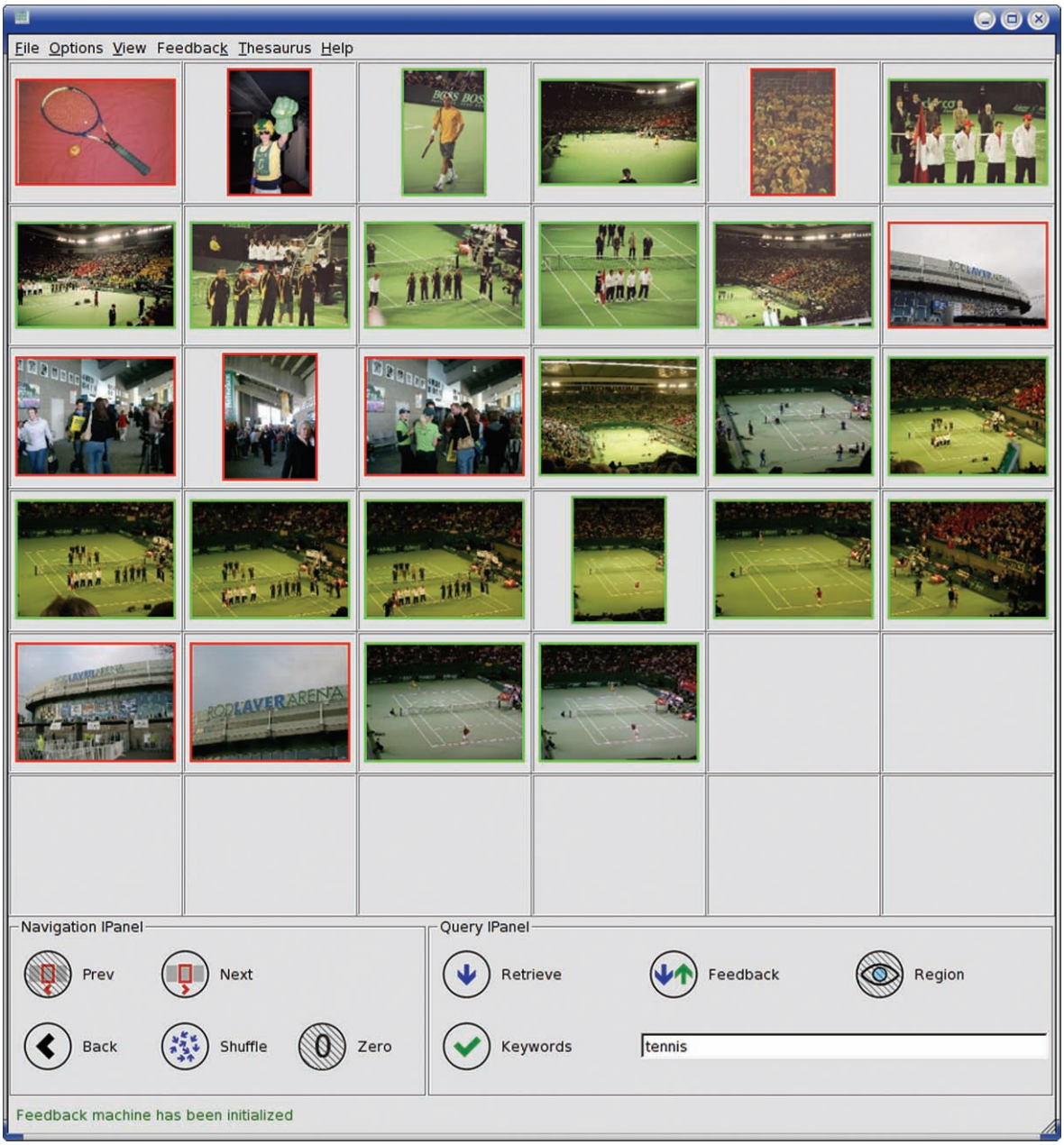
Two main learning algorithm families are used:

- *Generative*: the system tries to estimate density distribution of concepts in the visual space or other hidden variables [2]. Popular examples include Gaussian Mixture Models, Hidden Markov Models, Bayesian networks, Latent Semantic Analysis and translation models from the text processing community. The Expectation-Maximization algorithm is often used to train these models.
- *Discriminative*: instead of trying to model the distributions, discriminative approaches are focusing on detecting the boundaries between classes. For each visual concept, the annotation process is then often formalized as a two-class classification problem (present/not present). Although appearing to be a little bit more effective, discriminative approaches do not have the elegance of generative ones. They act more as black boxes and relationships between the different variables are not explicit, and thus difficult to analyze. The most famous algorithms are Support Vector Machine (SVM) [9,16], boosting (e.g., adaboost) [11] and all flavors of discriminant analysis (linear – LDA, biased – BDA, multiple – MDA, Fisher – FDA).

Both approaches may use global or local representations. Methods using bag-of-words representations are also called "multiple instance learning." Sometimes,

pre-processes may also be used to prepare the data in order to enhance the performances or to reduce the computational costs: feature selection, dimensionality reduction, scaling or normalization. Current systems are often composed of several components, using different low-level features, combining them according to different schemes and training models with multiple learning strategies.

Once the models have been learned, they can be used to predict the visual concepts. Two types of predictions are possible. Hard decision simply indicates the presence or absence of the concept. Soft decision also provides a degree of confidence in the prediction, allowing ranking more easily the results when answering an end-user query and thus improving the retrieval of pertinent images earlier.



Automatic Image Annotation. Figure 1. Pictures annotated with the "tennis" keyword.

Current Results

The different methods are actually mature enough to predict global visual concepts like image types and scene categories. Regarding local concepts, huge improvements still need to be made in order to provide useful applications to real users. Both dense sampling and point-of-interest have shown to perform quite well on research databases, but results on real databases are quite poor [9,12]. A good indication of state-of-the-art performances can be obtained in the results of official benchmark campaigns like ImagEval, Pascal VOC, Imageclef or Trecvid. In all cases, the contextual information (visual or from the existing metadata) has shown to be of great importance in the results.

When the availability of correctly annotated images for a given visual concept is not guaranteed, the offline

Automatic Image Annotation. Figure 2. Pictures displayed after two iterations.

learning approach is not possible. One of the solutions is then to interact with a user through relevance feedback, also called interactive learning. In a few iterations, the user will provide the system positive and negative examples and guide it to recognize the visual concept. Part of the mechanism involved are the same as offline learning, but the training labels are provided online by a user. As an example, the following screen captures from Ikona [3] are showing the IAPR-TC12 database, used for the Imageclef benchmark. The first screen displays all the images annotated with the "tennis" keyword. The user is only interested in pictures where a tennis court is visible. He indicates positive (green border) and negative (red border) examples. After two iterations, one can see on the second screen that a lot of tennis court pictures have been retrieved. None of them was annotated with the "tennis" keyword. After a few more iterations, when no more correct pictures are retrieved from the database, the user is able to annotate massively all the pictures gathered through the iterations that were kept in a specific basket (third screen) (Figs. 1–3.)

Key Issues and Future Research

The lack of generalization ability for both visual features and learning algorithms has to be compensated by a huge number of training examples. Depending on the complexity of the visual concept, a good estimation is around a hundred positive examples and ten times more negatives examples for the training set. Paradoxically, despite the tremendous amount of images available nowadays, finding content that has been reliably annotated for training dataset is hard.

The computational complexity is also still too high for real-time annotation when dealing with several thousands of visual concepts. Research is made on scalability issues in machine learning and is linked to existing high-dimensional data indexing structures.

Progresses for better description and integration of all types of available information need to be achieved.

There are also some questions arising: will the problem be solved with more computational power when one is able to process images at every scale and location in real-time? Are massive collaborative annotation websites, like Flickr, going to change the annotation paradigm

Automatic Image Annotation. Figure 3. All positive pictures basket allowing mass-annotation.

by transforming Internet in a giant common repository? What is the impact of GPS metadata, and more generally all the new information captured directly when a photograph is taken?

Key Applications

- Professional content owners: post-editing
- Personal family and holiday photo albums
- Web image search
- Searching into poorly human-made annotated corpora enhancing the quality of search results

Cross-references

- ▶ Annotation-Based Image Retrieval
- ▶ Boosting
- ▶ Content-based Image Retrieval (CBIR)
- ▶ Image Database
- ▶ Image Retrieval and Relevance Feedback
- ▶ Object Detection and Recognition
- ▶ Object Recognition
- ▶ Support Vector Machine
- ▶ Visual Content Analysis

Recommended Reading

1. Amores J., Sebe N., and Radeva P. Context-based object-class recognition and retrieval by generalized correlograms. IEEE Trans. Pattern Anal. Mach. Intell., 29(10):1818–1833, 2007.
2. Barnard K., Duygulu P., Forsyth D., de Freitas N., Blei D.M., and Jordan M.I. Matching words and pictures. J. Mach. Learn. Res., 3:1107–1135, 2003.
3. Boujemaa N., Fauqueur J., Ferecatu M., Fleuret F., Gouet V., Le Saux B., and Sahbi H. Ikona: interactive specific and generic image retrieval. In Proc. Int. Workshop on Multimedia Content-Based Indexing and Retrieval, 2001. Available at: http://www-rocg.inria.fr/imedia/mmcbirzod.html.
4. Boujemaa N., Fauqueur J., and Gouet V. What's beyond query by example? Technical report, INRIA, 2003.
5. Datta R., Li J., and Wang J.Z. Content-based image retrieval – approaches and trends of the new age. In Proc. 7th ACM SIGMM Int. Workshop on Multimedia Information Retrieval, 2005, pp. 253–262.
6. Enser P.G.B., Sandom C.J., and Lewis P.H. Automatic annotation of images from the practitioner perspective. In Proc. 4th Int. Conf. Image and Video Retrieval, 2005, pp. 497–506.
7. Hanjalic A., Sebe N., and Chang E. Multimedia content analysis, management and retrieval: trends and challenges. In Proc. SPIE: Multimedia Content Analysis, Management, and Retrieval, 2006.
8. Hare J.S., Lewis P.H., Enser P.G.B., and Sandom C.J. Mind the gap: another look at the problem of the semantic gap in image retrieval. In Proc. SPIE: Multimedia Content Analysis, Management, and Retrieval, 2006.
9. Hervé N. and Boujemaa N. Image annotation: which approach for realistic databases? In Proc. 6th ACM Int. Conf. Image and Video Retrieval, 2007, pp. 170–177.
10. Lew M.S., Sebe N., Djeraba C., and Jain R. Content-based multimedia information retrieval: State of the art and Challenges. ACM Trans. Multimedia Comp., Comm., and Appl., 2(1):1–19, 2006.
11. Opelt A., Pinz A., Fussenegger M., and Auer P. Generic object recognition with boosting. Pattern Anal. Mach. Intell., 28 (3):416–431, 2006.
12. Ponce J., Hebert M., Schmid C., and Zisserman A (eds.). Toward category-level object recognition. Springer-Verlag Lecture Notes in Computer Science, 2006.
13. Smeulders A.W.M., Worring M., Santini S., Gupta A., and Jain R. Content-based image retrieval at the end of the early years. IEEE Trans. Pattern Anal. Mach. Intell., 22(12):1349–1380, 2000.
14. Szummer M. and Picard R.W. Indoor-outdoor image classification. In Proc. Workshop on Content-based Access to Image and Video Databases, Bombay, 1998.
15. Vailaya A., Jain A., and Zhang H-J. On image classification: city images vs. landscapes. Pattern Recognit. J., 31(12):1921–1935, 1998.
16. Zhang J., Marszalek M., Lazebnik S., and Schmid C. Local features and kernels for classification of texture and object categories: a comprehensive study. Int. J. Comput. Vis., 73(2):213–238, 2007.

Automatic Induction

▶ Grammar Inference

Automatic Language Induction

▶ Grammar Inference

Automatic Wrapper Induction

▶ Fully Automatic Web Data Extraction

Autonomic Database Replica Allocation

▶ Autonomous Replication

Autonomic Database Systems

▶ Self-Management Technology in Databases

Autonomic Query Processing

► Adaptive Query Processing

Autonomous Message Queuing Systems

► Adaptive Middleware for Message Queuing Systems

Autonomous Replication

Cristiana Amza
University of Toronto, Toronto, ON, Canada

Synonyms

Database provisioning; Adaptive database replication; Autonomic database replica allocation

Definition

Autonomous database replication refers to dynamic allocation of servers to applications in shared server clusters, in such a way to meet per-application performance requirements. Autonomous database replication enables the service provider to efficiently multiplex data center resources across applications in order to save per-server costs related to human management, power and cooling.

Historical Background

The concept of Autonomic Computing and the associated research area of automated, adaptive self-management in data centers was introduced by IBM as a grand-challenge project in the early 2000s. Other companies, which have responded or have had similar proposals of their own include Microsoft, Intel, Sun and HP. Related industry efforts in this area have been on developing open standards for resource monitoring tools e.g., as available on IBM's Alphaworks, (http://www.alphaworks.ibm.com.) and academic or collaborative industry-academia efforts related to applying machine learning techniques for automated adaptation in cluster systems [6,8,13].

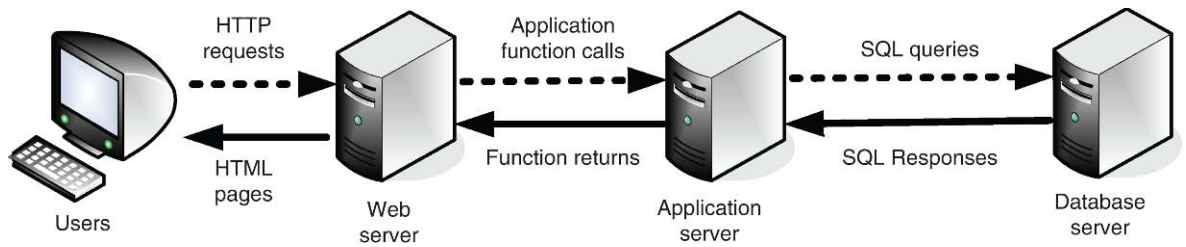
Foundations

Dynamic content servers, such as Amazon.com and eBay.com, commonly use a three-tier architecture (see Fig. 1) that consists of a front-end web server tier, an application server tier that implements the business logic, and a back-end database tier that stores the dynamic content of the site.

Large data centers may host multiple applications concurrently, such as e-commerce, auctions, news and games. The cooling and power costs of gross hardware over-provisioning for each application's estimated peak load are making efficient resource usage crucial. Furthermore, the excessive personnel costs involved in server management motivate an automated approach to resource allocation for applications in large sites.

Dynamic resource allocation techniques, i.e., dynamic provisioning of servers to multiple applications in each tier of the dynamic content site, have been recently introduced to address the increasing costs of ownership for large dynamic content server clusters. These automatic solutions add servers to an application's allocation based on perceived or predicted performance bottlenecks caused by either load spikes or component failures; they remove resources from a application's allocation when in underload.

If services experience daily patterns with peak loads for each service type at a different time (e.g., daytime for e-commerce, evening for auctions, morning for news sites, night for gaming), there are opportunities for re-assigning hardware resources from one service



Autonomous Replication. Figure 1. Three-tier architecture.

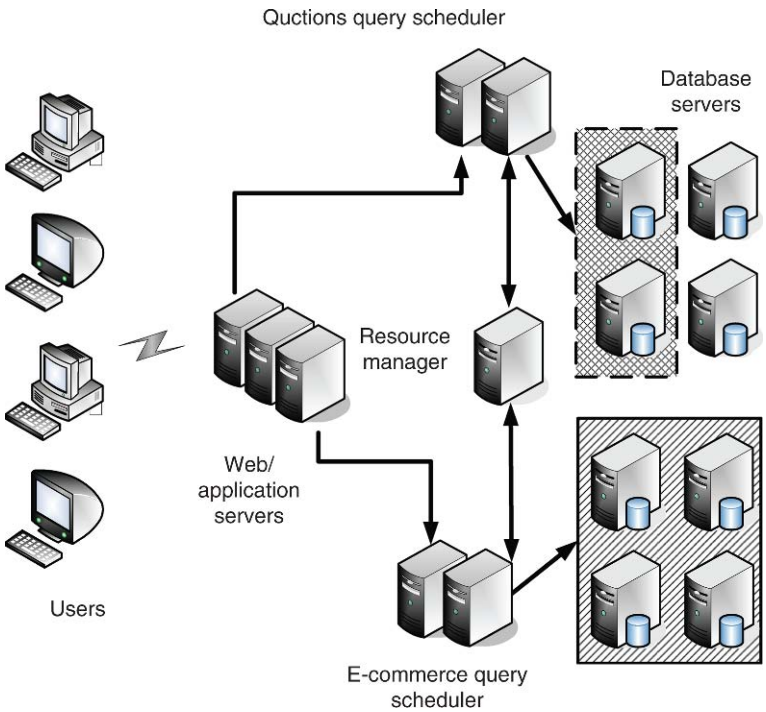
to another. Thus, instead of gross hardware over-provisioning for each application's estimated peak load, dynamic resource provisioning techniques enable the service provider to efficiently multiplex data center resources across applications.

Common Architecture for Database Replica Provisioning

Figure 2 shows the common architecture of sites with dynamic provisioning in the database server tier. A resource manager makes the replica allocation decisions for each application hosted on the site based on the application requirements and the current system state. The requirements are expressed in terms of a service level agreement (SLA) that consists of a latency requirement on the application's queries. The current system state includes the current performance of this application and the system capacity. The resource manager operates in two modes, underload and overload. During underload, the number of replicas exceeds overall demand and allocation decisions per application are made independently. During overload, the manager uses either a utility-based scheme, e.g., profit-based, or a fairness scheme, e.g., equal share, to allocate replicas to applications.

The allocation decisions are communicated to a set of schedulers, one per application, interposed between the application and the database tiers. Each scheduler fulfills the following functions. (i) It keeps track of the current *database set* allocated to its application and allocates or removes replicas from its managed set according to the resource manager's decisions. (ii) It distributes the corresponding incoming requests onto the respective database replicas. (iii) It periodically samples various system and application metrics, e.g., the average application latency from its *database set* in order to perceive or predict resource bottlenecks. (iv) It provides consistent replication, e.g., one-copy serializability [5], at all the replicas allocated to the application it manages.

Strong consistency is desirable for dynamic provisioning in the database tier of a multi-tier data center, for transparency reasons; the replicated nature of the database back-end and its configuration adaptations are thereby hidden from the application server, which interacts with the replicated back-end as with a single database. Any eager replication scheme with strong consistency guarantees, such as, 1-copy-serializability or 1-copy-snapshot-isolation



Autonomous Replication. Figure 2. Cluster architecture.

[5] can be used within each application's replica set. However, existing dynamic provisioning schemes typically leverage the presence of the scheduler and the synchronous, one query at a time, nature of the communication between the application and database tiers in a data center to implement a middleware replication solution in the scheduler itself. Upon receiving a query from the application server, the scheduler sends the query using a read-one, write-all replication scheme to the replica set allocated to the application. The scheduler assigns a global serialization order to all transactions pertaining to its application, e.g., based on conservatively-perceived table-level conflicts between transactions, and ensures that transactions execute in this order at all the corresponding database replicas. Table-level concurrency control affords optimizations based on conflict awareness in the scheduler [2,3], which offset any penalties due to the coarse-grain control for read-intensive e-commerce applications [11].

The scheduler is also in charge of bringing a new replica up to date by a process called *data migration*, during which all missing updates are applied on that replica. Integrating a stale replica into an application's allocation occurs through an on-line reconfiguration technique [10,11] without stalling on-going transactions on already active replicas for that application. Finally, each scheduler may itself be replicated for availability [2,3].

Overview of Dynamic Provisioning Solutions

Several fully-transparent provisioning solutions [4,9,12] have been recently introduced to address the increasing cost of management problem. Many of these approaches [4,9,12] investigate dynamic provisioning of resources within the (mostly) stateless web server and application server tiers. This entry focuses on the dynamic resource allocation solutions within the stateful database tier, which commonly becomes the bottleneck [1].

Regardless of the tier it applies to, a dynamic provisioning solution can be either *proactive* or *reactive* depending on its capabilities for predicting performance bottlenecks in the dynamic content server. *Proactive* solutions use sophisticated system models for prediction, such as, queuing models [4], utility models [12], machine learning models [6,8,13] or marketplace approaches [7]. *Proactive* provisioning techniques use the system model predictions for triggering allocations in advance of expected need. This is especially important for tiers with a higher adaptation delay, such as the database tier. In contrast, *reactive* approaches do not use prediction, but rather detect and react to existing resource bottlenecks. They rely on predefined thresholds for application-level or system metrics, such as latency or CPU usage for triggering changes in application allocation.

Challenges for Database Replica Provisioning

Adapting to a bottleneck caused by either a workload spike or a failure, by allocating additional replicas to the application, poses a set of challenges, which is summarized next.

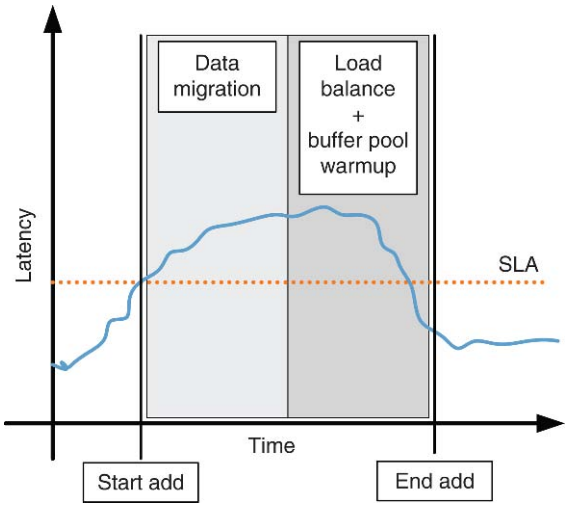
Adaptation Delay

Adding a database replica to an application's allocation is not an immediate process. The database state of the new replica(s) for that application will be stale and must be brought up-to-date, or a new instance of that application may need to be installed before it can be used. Furthermore, load balancing for the old and new replicas needs to occur and the buffer pool at the new replica(s) needs to be warm before the new replica(s) can be used effectively.

Oscillations in Allocation

Oscillations in database allocations to applications may occur during system instability induced by adaptations. As discussed earlier, the replica addition process can be long. *During* the adaptation phases, i.e., data migration, buffer pool warmup and load stabilization, the latency will remain high or may even temporarily continue to increase as shown in Figure 3. Latency sampling during this potentially long time is thus not necessarily reflective of a continued increase in load, but of system instability after an adaptation is triggered. If the system takes further decisions based on sampling latency during the stabilization time, it may continue to add further replicas which are unnecessary, hence will need to be removed later. This is an oscillation in allocation which carries performance penalties for other applications running on the system due to potential interference.

Both reactive and proactive policies that measure application-level metrics periodically, including during the replica addition process, can suffer from allocation instability. Allocation oscillations, in their turn, cause cross-application interference due to the



Autonomous Replication. Figure 3. Latency instability during replica addition.

price paid for warming up the buffer pool as part of the "context-switch" between applications on the machines involved.

While rapid load fluctuations may induce similar behavior, simple smoothing or filtering techniques can offer some protection to very brief load spikes. All existing dynamic provisioning schemes use some form of smoothing or filtering, to dampen brief load fluctuations.

Accurate and Lightweight Modeling

As previously mentioned proactive provisioning is desirable in the database back-end tier due to the higher adaptation delay inherent in this tier. The challenge for proactive database tier provisioning lies in accurately modeling the behavior of a replicated database back-end tier. Many factors, such as the wide range of query execution times typical in e-commerce workloads, the load balancing policy, caching effects, the replica consistency maintenance algorithm, etc. may influence performance. The derivation of a classic analytical model taking into account even a subset of these factors can be time consuming, hence unsuitable for on-line modeling and adaptation. Therefore, typically, pro-active provisioning schemes use machine learning approaches, such as K-Nearest-Neighbors (KNN) [6], or Support Vector Machine Regression (SVM) [8] to estimate performance models used for on-line adaptation. KNN-based approaches, or similar table-driven provisioning approaches [14] use off-line measurements and simple interpolation of response time values for various database server configurations and workloads. SVM-based approaches collect similar measurements[8] on-line to derive a regression function approximating the performance model dynamically.

Design Choices and Trade-offs for Database Provisioning

In designing a dynamic provisioning solution for the database back-end, it is necessary to consider the design trade-off between allocating replicas to applications in a *disjoint* versus *overlapping* manner. Consider the case where a *disjoint* set of machines is allocated to host the replicas of each application and dynamically adjusted to each application's cluster partition. When an application requires an additional replica, it must use an unallocated machine or a machine allocated to another application. In either case, the full *adaptation delay* for replica addition is incurred to the application.

Replica addition delay can be avoided altogether with *fully-overlapped* replicas, where all the database applications are replicated across all the available cluster machines. In this case, there is no replica addition delay because replicas do not have to be added or removed. However, this approach causes interference due to resource sharing. For example, when multiple database applications run on the same machine their performance can degrade due to buffer pool interference. This discussion shows that there is a trade-off between using disjoint and fully-overlapped replica allocation strategies. Disjoint allocation reduces interference and thus improves steady-state performance. Fully-overlapped allocation avoids replica addition delay and thus can speed up the system's response to load spikes and failures.

A compromise solution is to use a *partial overlap* strategy [11], where the application allocations are disjoint, but, for each application, batched updates are periodically executed on a set of database machines outside of that application's allocation, thus keeping them partially up-to-date.

Key Applications

Autonomic database replication is used for dynamic resource allocation in the database back-end of dynamic content web sites hosting e-commerce applications. Write queries in dynamic content applications are

typically more lightweight and have a much lower memory footprint compared to read queries [2]. For instance, in e-commerce applications, an update query typically updates only the record pertaining to a particular customer or product, while read queries caused by browsing involve expensive database joins as a result of complex search criteria. Moreover, read queries are much more frequent than write queries. Hence, a *partial overlap* replication solution, which causes minimal resource interference, is typically used in order to reduce the adaptation delay [11].

Cross-references

► Database Replication
► Replication for Scalability

Recommended Reading

1. Amza C., Chanda A., Cox A.L., Elnikety S., Gil R., Rajamani K., Zwaenepoel W., Cecchet E., and Marguerite J. Specification and implementation of dynamic web site benchmarks. In Proc. 5th IEEE Workshop on Workload Characterization, 2002, pp. 3–13.
2. Amza C., Cox A.L., and Zwaenepoel W. Conflict-aware scheduling for dynamic content applications. In Proc. 4th USENIX Symp. on Internet Tech. and Syst., 2003, pp. 6–6.
3. Amza C., Cox A.L., and Zwaenepoel W. Distributed versioning: consistent replication for scaling back-end databases of dynamic content web sites. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2003, pp. 282–304.
4. Bennani M.N. and Menasce D.A. Resource allocation for autonomic data centers using analytic performance models. In Proc. 2nd Int. Conf. on Autonomic Computing, 2005, pp. 229–240.
5. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison-Wesley, Reading, Massachusetts, 1987.
6. Chen J., Soundararajan G., and Amza C. Autonomic provisioning of backend databases in dynamic content web servers. In Proc. 3rd Int. Conf. on Autonomic Computing, 2006, pp. 123–133.
7. Coleman K., Norris J., Candea G., and Fox A. Oncall: defeating spikes with a free-market server cluster. In Proc. 1st Int. Conf. on Autonomic Computing, 2004, pp. 198–205.
8. Ghanbari S., Soundararajan G., Chen J., and Amza C. Adaptive learning of metric correlations for temperature-aware database provisioning. In Proc. 4th Int. Conf. on Autonomic Computing, 2007, pp. 26.
9. IBM Corporation. Automated provisioning of resources for data center environments. http://www-306.ibm.com/software/tivoli/solutions/provisioning/, 2003.
10. Liang W. and Kemme B. Online recovery in cluster databases. In Advances in Database Technology, Proc. 11th Int. Conf. on Extending Database Technology, 2008, pp. 121–132.
11. Soundararajan G. and Amza C. Reactive provisioning of backend databases in shared dynamic content server clusters. ACM Trans. Auton. Adapt. Syst, 1(2):151–188, 2006.
12. Tesauro G., Das R., Walsh W.E., and Kephart J.O. Utility-function-driven resource allocation in autonomic systems. In Proc. 2nd Int. Conf. on Autonomic Computing, 2005, pp. 342–343.
13. Tesauro G., Jong N.K., Das R., and Bennani M.N. On the use of hybrid reinforcement learning for autonomic resource allocation. Cluster Computing, 10(3):287–299, 2007.
14. Walsh W.E., Tesauro G., Kephart J.O., and Das R. Utility functions in autonomic systems. In Proc. 1st Int. Conf. on Autonomic Computing, 2004, pp. 70–77.

Autonomous Message-oriented Middleware

► Adaptive Middleware for Message Queuing Systems

Average Precision

Ethan Zhang[1,2], Yi Zhang[1]
[1]University of California, Santa Cruz, CA, USA
[2]Yahoo! Inc., Santa Clara, CA, USA

Definition

Average precision is a measure that combines recall and precision for ranked retrieval results. For one information need, the average precision is the mean of the precision scores after each relevant document is retrieved.

$$Average\ Precision = \frac{\sum_r P@r}{R}$$

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.

Key Points

The average precision is very sensitive to the ranking of retrieval results. The relevant documents that are ranked higher contribute more to the average than the relevant documents that are ranked lower. Changes to the ranking of relevant documents have a significant impact on the average precision score. Average precision is considered a reasonable

evaluation measure for emphasizing returning more relevant documents earlier.

Cross-references

- ▶ MAP
- ▶ Mean Average Precision
- ▶ Precision
- ▶ Precision at n
- ▶ P@n
- ▶ Recall
- ▶ Standard Effectiveness Measures

Average Precision at n

Nick Craswell, Stephen Robertson
Microsoft Research Cambridge, Cambridge, UK

Synonyms

AP@n

Definition

Average Precision at n is a variant of Average Precision (AP) where only the top n ranked documents are considered (please see the entry on Average Precision for its definition). AP is already a top-heavy measure, but has a recall component because it is normalized according to R, the number of relevant documents for a query. In AP@n there are a number of options for normalization, for example, normalize by n or normalize by min(n,R).

Key Points

The well-known measure Average Precision has a number of lesser-known variants, used in TREC [3] and elsewhere. Before and during TREC-1, it was usual to calculate an 11-point interpolated Precision-Recall curve, and take the average of these 11 precision values, giving an "interpolated AP." In TREC-2 and beyond, the modern non-interpolated AP was introduced. It calculates precision at each relevant document.

A number of other AP variants arise in a precision-oriented setting, where it is possible to calculate Average Precision at n. AP@n takes into account both the number of relevant documents in the top n and the positions of those documents. This is in contrast to Precision at n (P@n), which ignores position. It is defined as:

$$AP@n = \sum_{i=1}^{n} \frac{rel(i) \times P@i}{NF}$$

where rel(i) = 1 if the ith retrieved document is relevant and rel(i) = 0 otherwise, and NF is the normalization factor. In Average Precision it is usual to normalize by the number of relevant documents, i.e., NF = R. In a precision-oriented setting, this presents two problems. First, in precision-oriented evaluation one may not have judged enough documents to know R, or estimate it accurately. Second, if R is greater than n, a ceiling of n/R is imposed on the measure. For example, if one knows R = 100 relevant documents, then the best possible top-20, containing 20 relevant documents, will score an AP@20 of 0.2.

Three alternate normalization factors (NF) have been considered, only one of which has been used in TREC. Here, r is the number of relevant documents retrieved and n is the number of documents retrieved.

- *Normalize by r:* Baeza-Yates and Ribeiro-Neto [1] includes this variant, calling it "Average Precision at Seen Relevant Documents." This measure has the property that it may decrease when a relevant document is promoted into the top-n, because this increases the normalization factor. This seems counter-intuitive.
- *Normalize by n:* This variant was introduced for use in Web search evaluation [2]. In cases where R is less than n, this variant has a ceiling of less than 1. The scale of the measure is $0 \leq$ AP@n $\leq$ min (R/n,1). It may be considered undesirable that the scale of the measure varies from query to query, for example when measuring the mean.
- *Normalize by min(n,R):* This is the "modified average precision" from the TREC-7 Very Large Collection track [3]. The ceiling is always 1 for this variant.

The normalization NF = min(n,R) allows AP@n scores to have the full range of 0.1, while retaining the property that promoting a relevant document always increases the score.

The arithmetic mean of AP@n over a set of queries can be called Mean Average Precision at n (MAP@n), in the same way that Average Precision relates to Mean Average Precision.

Cross-references

- ▶ Average Precision
- ▶ Mean Average Precision
- ▶ Precision at n
- ▶ Precision-Oriented Effectiveness Measures

Recommended Reading

1. Baeza-Yates R.A. and Ribeiro-Neto B. Modern Information Retrieval. Addison-Wesley, Reading, MA, 1999.
2. Hawking D., Craswell N., Bailey P., and Griffiths K. Measuring search engine quality. Inf. Retr., 4(1):33–59, 2001.
3. Voorhees E.M. and Harman D.K. TREC: Experiment and Evaluation in Information Retrieval. MIT Press, Cambridge, MA, 2005.

Average Precision Histogram

Steven M. Beitzel[1], Eric C. Jensen[2], Ophir Frieder[3]
[1]Telcordia Technologies, Piscataway, NJ, USA
[2]Twitter, Inc., San Fransisco, CA, USA
[3]Georgetown University, Washington, DC, USA

Definition

The average precision histogram plots the performance of a single run produced by an information retrieval system on a per-query basis. Each data point on the abscissa represents a query used in the evaluation process. The corresponding points on the ordinate measures the difference in this run's performance on the given topic relative to the median average precision of other runs on that topic.

Key Points

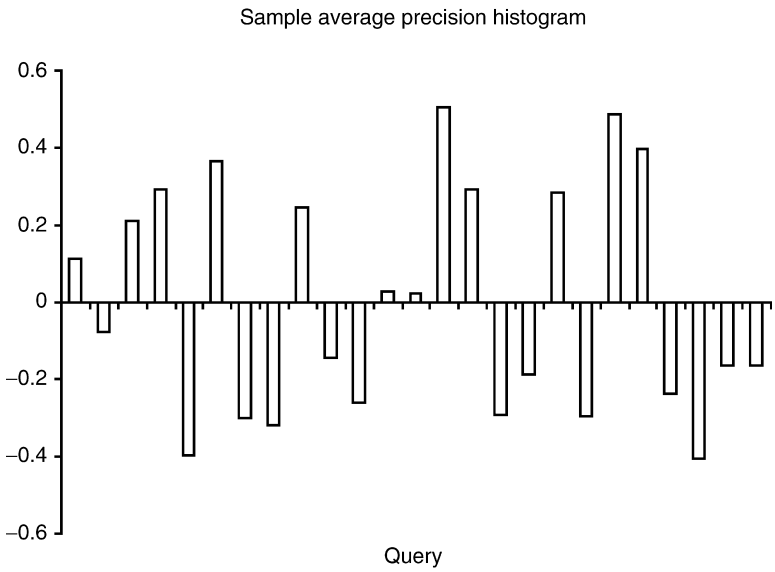
Average precision histograms are often used to illustrate the performance differences that an information retrieval system may exhibit across different queries in an evaluation set in relation to other systems or runs that are evaluated on the same set of queries. These histograms can be used to identify the queries in an evaluation set which pose the most difficulty for information retrieval systems. Various tracks in the NIST Text Retrieval Conference (TREC) have used average precision histograms in the post-competition analysis of submitted runs [1]. An example average precision histogram for 25 queries is shown in Fig. 1.

Cross-references

- ▶ Average Precision
- ▶ Effectiveness Involving Multiple Queries
- ▶ Geometric Mean Average Precision
- ▶ Mean Average Precision

Recommended Reading

1. National Institute of Standards and Technology. TREC-2003 Common Evaluation Metrics. Available online at: http://trec.nist.gov/pubs/trec12/appendices/measures.ps (retrieved on August 27, 2007), 2003.



Average Precision Histogram. Figure 1. Example average precision histogram.

A

Average R-Precision

Steven M. Beitzel[1], Eric C. Jensen[2], Ophir Frieder[3]
[1]Telcordia Technologies, Piscataway, NJ, USA
[2]Twitter, Inc., San Fransisco, CA, USA
[3]Georgetown University, Washington, DC, USA

Definition

The Average R-precision is the arithmetic mean of the R-precision values for an information retrieval system over a set of n query topics. It can be expressed as follows:

$$ARP = \frac{1}{n}\sum_{n} RP_n$$

where RP represents the R-Precision value for a given topic from the evaluation set of n topics. R-Precision is defined as the precision after R documents have been retrieved by the system, where R is also the total number of judged relevant documents for the given topic. Precision is defined as the portion of retrieved documents that are truly relevant to the given query topic.

Key Points

R-precision places lower emphasis on the exact ranking of the relevant documents returned by an information retrieval system. This can be useful when a topic has a large number of judged relevant documents, or when an evaluator is more interested in measuring aggregate performance as opposed to the fine-grained quality of the ranking provided by the system.

As an example, consider two query topics: topic A has ten relevant documents, and topic B has six relevant documents. Suppose further that an information retrieval system returns five relevant documents in the top ten retrieved for topic A, two relevant documents in the top six retrieved for topic B. For this case, Average R-precision for this run would be:

$$ARP = \frac{\frac{5}{10} + \frac{2}{6}}{2} \approx 0.4167$$

Cross-references

- ▶ Effectiveness Involving Multiple Queries
- ▶ Mean Average Precision
- ▶ R-Precision

Recommended Reading

1. National Institute of Standards and Technology. TREC-2004 Common Evaluation Measures. Available online at: http://trec.nist.gov/pubs/trec14/appendices/CE.MEASURES05.pdf (retrieved on August 27, 2007), 2005.

AXML

- ▶ Active XML