P

P/FDM

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

Definition

P/FDM [5–7] integrated a functional data model with the logic programming language Prolog for general-purpose computation. The data model can be seen as an Entity-Relationship diagram with sub-types, much like a UML Class Diagram. The idea was for the user to be able to define a computation over objects in the diagram, instead of just using it as a schema design aid. Later versions of P/FDM included a graphic interface [2,4] to build queries in DAPLEX syntax by clicking on the diagram and filling in values from menus.

P/FDM was subsequently extended with constraints [3] and with alternative back-ends to remote databases [6], in the spirit of the original MULTIBASE system.

P/FDM is a vehicle to test a system designed on the principle of Data Independence, whereby Functions represent computations that are expressed in a way that is completely independent of data storage (arrays, lists of objects, indexed files etc.). Functions can be sent across the internet and applied to data in a different form, which was very useful for federated data.

Key Points

In P/FDM, the DAPLEX language was deliberately altered from its original specification so that its semantics could be defined by equivalent *Comprehensions* [1]. In particular, simple assignment operations, if present, could only take place within the innermost loop. Based on this, a very successful early optimizer was written in Prolog by Paton, and used in several bioinformatics applications [6,8].

In P/FDM, data independence is ensured by using a small sparse set of built-in predicates *getentity (entity-type, key, instance-variable)* and *getfunctionval (function-name, argument-variable, value-variable)*. The first of these predicates requires that abstract entities be identifiable by unique, possibly compound, scalar keys [7]. The keys help with object identity and are very important when accessing or loading bulk data. This is a feature first used in ADAPLEX and EFDM, though P/FDM extends it to allow composed (single-valued) functions, which aids in forming hierarchical keys.

P/FDM queries are written in DAPLEX and translated into Prolog for evaluation. Backtracking in Prolog is used to give the effect of lazy evaluation in a functional programming language, accessing tuples or objects on demand. For example, assuming the following P/FDM declarations:

```
declare student ->> entity
declare name(student) -> string
key_of student is name

declare course ->> entity
declare cname(course) -> string
key_of course is cname
declare attends(student) ->> course
```

to print all the courses attended by Fred Jones, the DAPLEX query is:

```
for each F in student such that name
(F)="Fred Jones"
   for each C in attends(F)
   print(cname(C));
```

which generates the following Prolog:

```
getentity(student,'Fred Jones',F),
getfunctionval(attends,F,C),
getfunctionval(cname,C,N), write(N),
fail; true.
```

Cross-references

▶ Comprehensions
▶ Federated Database

► Functional Query Language
► Query Languages and Evaluation Techniques for Biological Sequence Data

Recommended Reading

1. Embury S.M. User Manual for P/FDM V.9.1. Technical report, Dept. of Computing Science, University of Aberdeen, 1995.
2. Gil I., Gray P.M.D., and Kemp G.J.L. A Visual Interface and Navigator for the P/FDM Object Database. In Proc. User Interfaces to Data Intensive Systems, 1999, pp. 54–63.
3. Gray P.M.D., Embury S.M., Hui K.Y., and Kemp G.J.L. The evolving role of constraints in the functional data model. J. Intell. Inform. Syst., 12:113–137, 1999.
4. Gray P.M.D. and Kemp G.J.L. Capturing quantified constraints in FOL, through interaction with a relationship graph. In Proc. 15th Int. Conf. Knowledge Eng. and Knowledge Management: Ontologies and the Semantic Web, 2006, pp. 19–26.
5. Gray P.M.D., Moffat D.S., and Paton N.W. A Prolog interface to a Functional Data Model database. In Advances in Database Technology, Proc. 1st Int. Conf. on Extending Database Technology, 1988, pp. 34–48.
6. Kemp G.J.L., Dupont J., and Gray P.M.D. Using the functional data model to integrate distributed biological data sources. In Proc. 8th Int. Conf. on Scientific and Statistical Database Management, 1996, pp. 176–185.
7. Paton N.W. and Gray P.M.D. Identification of database objects by key. In Proc. 2nd Int. Workshop on Object-Oriented Database Systems. LNCS 334. Springer, 1988, pp. 280–285.
8. Paton N.W. and Gray P.M.D. Optimising and executing daplex queries using prolog. Comput. J., 33(6):547–555, 1990.

P@n

► Precision at n

P2P Database

► Structured Data in Peer-to-Peer Systems

Page Cache

► Buffer Pool

Page Locking

► Concurrency Control – Traditional Approaches

Page Model

► Transaction Models – The Read/Write Approach

Page Representations

► Document Representations

Paging in Web Search Engines

► Web Search Result Caching and Prefetching

PAM (Partitioning Around Medoids)

► K-Means and K-Medoids

Parallel and Distributed Data Warehouses

Todd Eavis
Concordia University, Montreal, QC, Canada

Synonyms

Scalable decision support systems High performance data warehousing

Definition

To support the burgeoning data volumes now encountered in decision support environments, *parallel and distributed data warehouses* are being deployed with greater frequency. Having evolved from haphazard and often poorly understood repositories of operational information, the data warehouse itself has become one of the cornerstones of corporate IT architectures. However, as the underlying operational databases grow in size and complexity, so too do the associated data

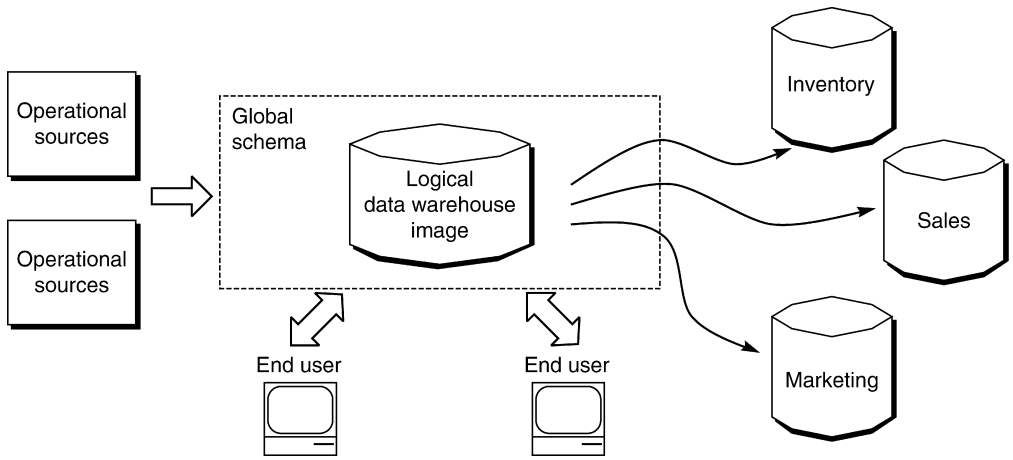
warehouses. In fact, it is not unusual for many corporate or scientific repositories to exceed a terabyte in size, with the largest now reaching 100 TB or more. While processing power has grown significantly during the past decade, the sheer scale of the workload places enormous strain on single CPU data warehousing servers. As a result, some form of data and/or query distribution is often employed in production environments. It is important to note, however, that while contemporary data warehouses are almost always based upon relational DBMS platforms, the unique characteristics and requirements of data warehouse environments often suggest design and optimization choices that are not often employed with general purpose parallelized database systems.

Historical Background

The terms "parallel DW" and "distributed DW" are very often used interchangeably. In practice, however, the distinction between the two has historically been quite significant. Distributed DWs, much like distributed DBs, grew out of a need to place processing logic and data in close proximity to the users who might be utilizing them. In general, multi-location organizations consist of a small number of distinct sites, each typically associated with a subset of the information contained in the global data pool. In the data warehousing context, this has traditionally lead to the development of some form of *federated* architecture. In contrast to monolithic, centralized DWs, federated models are usually constructed as a cooperative coalition of departmental or process specific *data marts*. A simple example is illustrated in Fig. 1. For the most part, design and implementation issues in such environments are similar to those of distributed operational DBMSs [11]. For example, it is important to provide a single transparent conceptual model for the distinct sites and to distribute data so as to reduce the effects of network latency and bandwidth limitations. One unique feature of the distributed data warehouse environment is perhaps the emphasis on integration and consolidation of distributed operational data, a process known in DW terminology as Extract, Transform, and Load (ETL).

With respect to parallelism, research has again been influenced by parallel DBMS projects such as Gamma [3] that were initiated in the mid-to-late 1980s. By the 1990s, it had become clear that commodity-based "shared nothing" databases provided significant advantages over the earlier SMP (Symmetric Multi-Processor) architectures in terms of cost and scalability [4]. Subsequent research therefore focused on partitioning and replication models for the tables of the parallelized DBMS [13]. In general, researchers identified the importance of full p-way horizontal striping for large database tables.

Data warehouse researchers have continued to explore the issues related to table partitioning. In addition to complete p-way partitioning schemes, full or partial replication (i.e., duplication) of fragments has been investigated [12]. This technique has been further extended by virtualizing partial fragments over



Parallel and Distributed Data Warehouses. Figure 1. A simple federated model illustrating the mapping of a logical global schema on to a series of physically independent data marts.

physically replicated tables [7]. Typically, recent research in the area of table partitioning has exploited the use of "DBMS clusters." Here, rather than constructing a complete, parallel DBMS platform, the parallel system is essentially constructed as a series of commodity DBMS systems, "glued together" with a thin partition-aware wrapper layer.

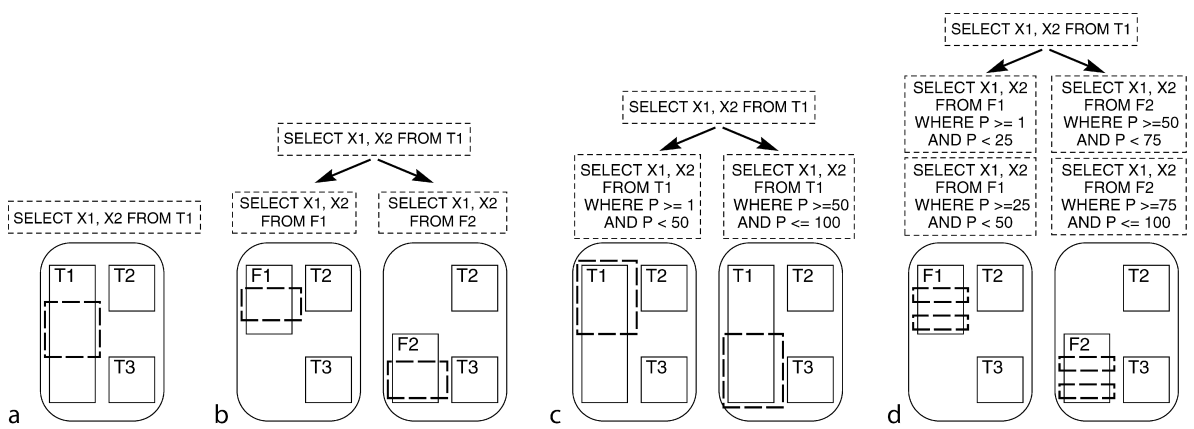
In addition to the continuation of the traditional partitioning work, a second important theme has been the parallelization of state-of-the-art sequential *data cube* generation methods. The data cube has become the primary abstraction for the multi-dimensional analysis that is central to modern data warehousing systems. Cube parallelization efforts have, in fact, taken two forms, one based upon data that is physically represented as array-based storage [8] and the other based upon relational storage [9,2]. In both cases, the complexity of the algorithm and implementation issues has lead to the development of relatively complete DBMS prototypes.

Foundations

Central to data warehousing systems is a denormalized logical *multi-dimensional model* known as the *Star Schema* (the normalized version is referred to as a Snowflake). A *Star Schema* consists of a single, very large *fact* table housing the measurement records associated with a given organizational process. During query processing, this fact table is joined to one or more *dimension* tables, each consisting of a relatively small number of records that define specific business entities (e.g., customer, product, store). A complete data warehouse typically consists of multiple such *Star Schema* designs.

While the *Star Schema* forms the basis of the relational data warehouse, it can be extremely expensive to query the fact table directly, given that it often consists of tens of millions of records or more. Typically, the basic *Star Schema* is augmented with compact, precomputed aggregates (called *group-bys* or *cuboids*) that can be queried much more efficiently at runtime. This collection of aggregates is known as the data *cube*. Specifically, for a d-dimensional space, $\{A_1, A_2,...,A_d\}$, the *cube* defines the aggregation of the 2^d unique *dimension* combinations across one or more relevant *measure* attributes. In practice, the generation and manipulation of the data *cube* is often performed by a dedicated *OLAP* (online analytical processing) server that runs on top of the underlying relational data warehouse. While the *OLAP* server may utilize either array-based (MOLAP) or table-based (ROLAP) storage, both provide the same, intuitive multi-dimensional representation for the end user.

Given the enormous size of these new data warehouses, some form of parallelism is often employed in production environments. One option is a "shared everything" architecture. Here, designers would likely employ a CC-NUMA system (Cache Coherent Non Uniform Memory Access) that supports a single global memory pool and some form of single virtual disk



Parallel and Distributed Data Warehouses. Figure 2. A three dimensional OLAP space showing all 2^d cuboids and the parent-child relationships between them. Each cell would contain a Total Sales aggregate value.

(e.g., a disk array). Such systems have the advantage that they are relatively easy to administer as the hardware transparently performs much of the "magic." That being said, shared everything designs also tend to be quite expensive and have limited scalability in terms of both the CPU count and the number of available disk heads. In terabyte-scale data warehouse environments, either or both of these constraints might represent a serious performance limitation.

For this reason, many vendors and researchers have turned towards distributed, "shared nothing" platforms for high performance database applications. In fact, the characteristics of DW processing environments make such models particulary attractive. The key distinctions between operational and DW processing include:

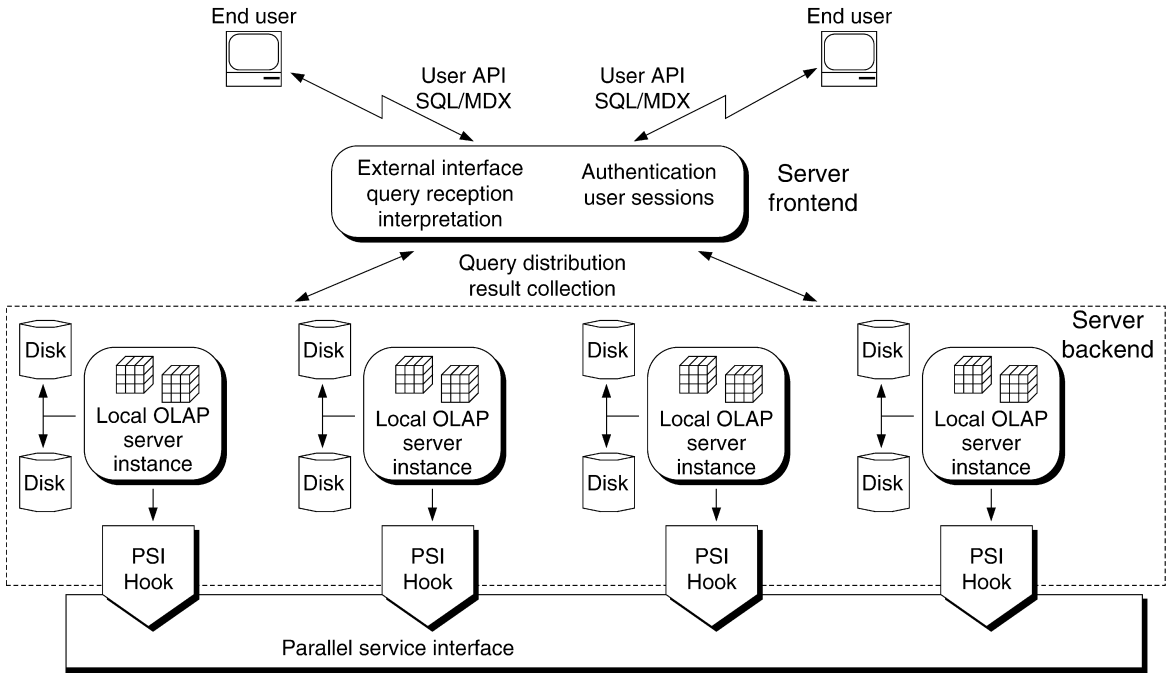
1. Operational systems tend to have high volume query streams. In contrast, DW systems typically process a much smaller number of user queries.
2. Operational queries have *high selectivity*, meaning that they touch relatively few records. Conversely, DW queries are comprehensive in scope, leading to very low selectivity and high I/O and computational load.

Why does this bode well for the use of shared nothing architectures? In large operational environments, high performance can often be achieved through the exploitation of what is often termed a "high throughout engine." Here, improved performance is obtained via *inter-query* parallelism; that is, a stream of small queries is executed concurrently across the parallel/distributed system. However, the low volume/low selectivity combination in DW environments generally argues against the use of inter-query parallelism since such a division of work would likely lead to extensive disk thrashing (i.e., resource conflict). Note, as well, that the use of indexes, a staple in operational systems, is often of relatively little value in DW environments since full table scans are generally more cost effective for low selectivity queries.

As a consequence, the fully distributed, "shared nothing" DBMS model has been particulary attractive for high performance DW practitioners. By partitioning the core fact tables across a large number of independently controlled CPU/disk combinations, shared nothing designs are ideally suited to the exploitation of *intra-query parallelism*. In this case, an individual query q is decomposed and simultaneously executed across the p nodes of the system, with each *partial query* running against approximately $1/p$ records of the partitioned fact table (the small dimension tables are typically replicated on each node). Figure 3b provides a simple example of this technique, contrasting it with the non-partitioned model typically utilized on a single node server (Fig. 3a). Though merging of results may be necessary, it is important to note that while the input partitions may be massive, the output of user-directed analytical DW queries is typically quitesmall. As a result, it should be seen that shared nothing data warehousing excels precisely because it offers tremendous I/O performance while simultaneously requiring only modest inter-node communication [10].

It is against this backdrop that many recent data warehousing partitioning projects have been set. Essentially, there are three forms of DW partitioning. In the first case, the fact tables are physically partitioned in the manner just described. While the performance with this approach can be impressive [15], it is also true that imbalances caused by inherent data skew make effective *a priori* data striping quite challenging. In response, the *virtual* partition model was proposed [1]. Here, as illustrated in Fig. 3c, the fact tables are replicated in full across the nodes of the DBMS cluster. Queries are then decomposed into sub-queries ($1/p$ records per node) that are run against virtual partitions mapped on top of the fully replicated tables. The advantage is that sub-queries may be (i) run against any cluster node and (ii) dynamically migrated to underutilized nodes. The downside is (i) the storage requirement associated with p copies of the primary fact table and (ii) the fact that the commodity DBMS will invoke a full table scan if the partitions are too large. The third partitioning method attempts to combine the best features of the previous methods. In *adaptive virtual partitioning* (AVP) [7], small virtual partitions are layered on top of larger physical partitions. The AVP algorithm dynamically determines the maximum sub-query size that does not invoke a table scan by iteratively probing the commodity DBMS with successively larger queries. The result is a set of re-locatable sub-queries that can generally be answered efficiently without access to the code base of the commodity DBMS systems. A simple illustration is provided in Fig. 3d.

While partitioning is the cornerstone of contemporary high performance data warehouses, it should be clear that such methods represent a sort of "brute



Parallel and Distributed Data Warehouses. Figure 3. Fundamental DW partitioning schemes. (a) Query executed on a single node server that houses a large fact table (T1) and two small dimension tables (T2, T3). (b) Physical partitioning, creating fact table *fragments* (F1, F2). (c) Virtual partitioning on the *clustering attribute* **P** of the replicated fact table. (d) Adaptive virtual partitioning, using fragments and multiple sub-queries.

force" approach to query resolution. Though the response time may indeed represent close to linear speedup as a function of CPU/disk count, the implicit assumption is that full processing of the atomic, transactional data is necessary. This is certainly true for arbitrary or ad-hoc user queries since nothing is known about the possible query format. However, most data warehouses support user interaction through some form of *OLAP* interface. At its core, *OLAP* represents an analytical environment for the intuitive manipulation of the data *cube*. As noted above, there are $O(2^d)$ such views or cuboids. By materializing some (or occasionally all) of these pre-aggregated views, DW/*OLAP* systems can dramatically improve run-time performance on common queries without resorting to massive fact table scans. The trade-off, of course, is that aggregates must be accurately maintained (i.e., updated). In fact, this requirement exerts considerable pressure on the underlying server as data volumes explode in size while, at the same time, update *windows* shrink. It is important to understand, however, that the run-time performance benefit resulting from cube materialization is likely to be significantly larger than the (at best) linear speedup achieved by a conventional parallel query engine. As such, parallel resources may be more valuable when utilized for the construction and ongoing maintenance of the *OLAP* data structures (e.g., summaries), rather than for brute force query execution.

Because of the extensive computational requirements of *cube* generation in large data warehouses, a number of parallel *cube* construction and querying architectures have in fact been proposed. Note the use of the word "architectures" to indicate comprehensive models that include computation, memory management, and physical storage. In the parallel environment, both array-based (MOLAP) and relational (ROLAP) systems have been explored. With respect to MOLAP parallelism, a distributed memory framework targeting IBM's SP2 was described in [8]. Here, cube construction begins by first equi-partitioning on a single attribute A_1. Using sequential MOLAP

techniques, all partial cuboids containing A_1 (exactly $\frac{2^d}{2}$ of the total) are constructed independently on each node. The data set is then re-partitioned on A_2 and the process is repeated. In total, d such rounds are required. The MOLAP mechanism is attractive in the parallel environment as it is well suited to the shared nothing model. Moreover, the parallelized MOLAP cuboids have the potential to provide extremely fast run-time performance (as is generally the case with MOLAP). Having said that, like all MOLAP systems, the sparsity of array-based storage does necessitate complex sparse array compression. In addition, data skew has the potential to generate significant load imbalance during each of the d re-partitioning phases.

Alternatively, parallelism can be exploited in purely relational environments. The cgmCube project, for example, also utilizes state-of-the-art sequential cube construction methods but materializes results in the form of conventional relational tables [2]. Both shared nothing and shared disk platforms are targeted. In this case, the *cube lattice* is physically materialized by first creating a weighted minimum cost spanning tree representation that identifies the most cost effective means by which to materialize all $O(2^d)$ cuboids. Parallelism is supported by decomposing the spanning tree into $p$sub-trees (using a k-min-max graph partitioning algorithm), each of which is computed in its entirety on a single node. The advantage here is that global costing decisions can be employed so as to significantly improve load balancing and to eliminate communication costs (note that the final cuboids can be re-partitioned in any way once the construction algorithm has terminated). Unlike array-based MOLAP systems, ROLAP does not provide implicit indexing, a problem of some significance given that traditional "single dimension" indexes such as the *b-tree* do not work particularly well in multi-dimensional spaces. cgmCube addresses this shortcoming by adding a forest of fully parallelized, multi-dimensional R-tree indexes. Ultimately, cgmCube's algorithmic components are integrated into a fully parallelized server prototype called Sidera that essentially functions as a federation of single node *OLAP* servers [5]. In addition to cube generation and indexing, Sidera adds functionality for caching, selectivity estimation, and the management of *dimensionhierarchies*, as well as fully parallelized sorting and aggregation support. A logical picture of the Sidera server is illustrated in Fig. 3. It should be noted that due to the enormous complexity of parallelized OLAP systems, no direct performance comparison of the MOLAP and ROLAP alternatives has ever been performed.

Finally, one should be aware that commercial parallel DW implementations are also available. Of particular interest in the current context are the dedicated *appliance* style architectures. Here, an integrated, parallel shared nothing hardware/software bundle is tailored specifically to data warehousing workloads and query patterns. Recently, the traditional "high end" DW system provider, Teradata, has been joined by new vendors such as Netezza, DATAllegro, and Greenplum that build upon commodity hardware and open source DBMS software. While each provides proprietary mechanisms for elements such as indexing and partitioning, the DW appliance vendors tend to rely primarily on the aforementioned "brute force" style of parallelism. At present, it does not appear that there is a direct movement of theoretical results from the research community to the industrial sector.

Key Applications

A quick review of the recent database literature indicates that the majority of the current work in high performance databases is actually associated with data warehousing. This should perhaps not be surprising since data warehouses represent some of the largest databases in existence today. In fact, with the emergence of the Internet as a vehicle for both data collection and distribution, one can only expect this trend to continue. So as the *average* size of production DWs pushes past the terabyte limit, parallelism is likely to become an increasingly common theme.

Future Directions

In the short term, one would expect to see continued interest in the exploitation of commodity DBMS systems within loosely federated parallel DBMS clusters. This approach makes a great deal of sense given the considerable maturity of such platforms. Beyond this, OLAP query performance might be further improved by investigating the parallelization of more recent sequential cube methods. The tree-based Dwarf Cube is an obvious target in this regard [14]. We can also expect to see a greater focus on the emerging trend of real time or *near* real time data warehousing, particulary for parallel systems that target large, dynamic data environments.

In the longer term, one possible area for further investigation is the convergence of parallel/distributed data warehouses and the new data centric Grid technologies [6]. To this point in time, the Grid framework has primarily been associated with the concurrent processing of "flat files" rather than highly structured databases. Nevertheless, the Grid model offers interesting opportunities for the distributed, secure, and equitable processing of publicly accessible data repositories.

Experimental Results

Virtually all of the research mentioned in this discussion has relevant experimental evaluations within the associated references. In general, the parallel implementations described above – both partition oriented and OLAP-based – are capable of achieving near linear speedup on contemporary shared nothing parallel systems of 8–32 nodes.

Cross-references

- ▶ Cube
- ▶ Cube Implementations
- ▶ Cube Lattice
- ▶ Data Warehousing Systems: Foundations and Architectures
- ▶ Dimension
- ▶ Measure
- ▶ Multidimensional Modeling
- ▶ OLAP
- ▶ On-Line Analytical Processing
- ▶ Optimization and Tuning in Data Warehouses
- ▶ Query Processing in Data Warehouses
- ▶ Star Schema

Recommended Reading

1. Akal F., Böhm K., and Schek H.-J. OLAP query evaluation in a database cluster: a performance study on intra-query parallelism. In Proc. 6th East European Conf. Advances in Database and Information Systems, 2002, pp. 218–231.
2. Dehne F., Eavis T., and Rau-Chaplin A. The cgmCUBE project: optimizing parallel data cube generation for ROLAP. J. Distr. Parallel Databases, 19(1):29–62, 2006.
3. DeWitt D., Ghandeharizadeh S., Schneider D., Bricker A., Hsaio H., and Rasmussen R. The gamma database machine project. Trans. Knowl. Data Eng., 2(1):44–62, 1990.
4. DeWitt D. and Gray J. Parallel database systems: the future of high performance database systems. Commun. ACM, 35(6):85–98, 1992.
5. Eavis T., Dimitrov G., Dimitrov I., Cueva D., Lopez A., and Taleb A. Sidera: a cluster-based server for online analytical processing. In Proc. Int. Conf. on Grid Computing, High-Performance, and Distributed Applications, 2007.
6. Fiser B., Onan U., Elsayed I., Brezany P., and Tjoa A.M. On-line analytical processing on large databases managed by computational grids. In Proc. 15th Int. Conf. Database and Expert Syst. Appl., 2004, pp. 556–560.
7. Furtado C., Lima A., Pacitti E., Valduriez P., and Mattoso M. Physical and virtual partitioning in OLAP database clusters. In Proc. Int. Symp. on Computer Architecture and High Performance Computing, 2005, pp. 143–150.
8. Goil S. and Choudhary A. High performance multidimensional analysis of large datasets. In Proc. 1st ACM Int. Workshop on Data Warehousing and OLAP, 1998, pp. 34–39.
9. Jin R., Vaidyanathan K., Yang G., and Agrawal G. Communication and memory optimal parallel data cube construction. IEEE Trans. Parallel Distr Syst., 16(12):1105–1119, 2005.
10. Morse S. and Isaac D. Parallel Systems in the Data Warehouse. Prentice-Hall, Englewood Cliffs, 1998.
11. Özsu M.T. and Valduriez P. Principles of distributed database systems 2nd edn. Prentice-Hall, Englewood Cliffs, NJ, 1999.
12. Röhm U., Böhm K., and Schek H.-J. Routing and physical design in a database cluster. In Advances in Database Technology, Proc. 7th Int. Conf. on Extending Database Technology, 2000, pp. 254–268.
13. Scheuermann P., Weikum G., and Zabback P. Data partitioning and load balancing in parallel disk systems. VLDB J., 7(1):48–66, 1998.
14. Sismanis Y., Deligiannakis A., Roussopoulos N., and Kotidis Y. Dwarf: shrinking the PetaCube. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 464–475.
15. Stohr T., Märtens H., and Rahm E. Multi-dimensional database allocation for parallel data warehouses. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 273–284.

Parallel Axes

- ▶ Parallel Coordinates

Parallel Coordinates

Alfred Inselberg[1,2]
[1]Tel Aviv University, Tel Aviv, Israel
[2]University of California-San Diego, La Jolla, CA, USA

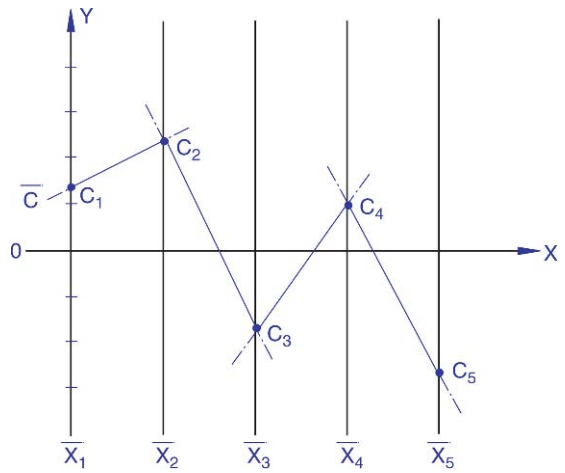
Synonyms

Parallel axes; Parallel Coordinates System (PCS); ||-coords; Multidimensional visualization; Parallel Coordinates Plot (PCP)

Definition

In the plane with *xy*-Cartesian coordinates N copies of the real line labeled $\bar{X}_1$, $\bar{X}_2$,...,$\bar{X}_N$ are placed equidistant and perpendicular to the *x*-axis. They are the axes of the multidimensional system of *Parallel Coordinates* all having the same positive orientation as the *y*-axis. An N-tuple, N-dimensional point, $C = (c_1, c_2,...,c_N)$ is represented by the polygonal line $\bar{C}$ whose N vertices are at the c_i values on each $\bar{X}_i$-axis as shown in Fig. 1. In this way, a 1–1 correspondence between points in N-dimensional space and polygonal lines with vertices on the parallel axes is established. In principle, a large number of axes can be placed and be seen parallel to each other. The representation of points is deceptively simple and much development with additional ideas is needed to enable the visualization of *multivariate relations* or equivalently multidimensional objects.

A dataset with *M* items has 2^M subsets anyone of which may be the one wanted. With a good data display the fantastic human pattern-recognition can not only cut great swaths searching through this combinatorial explosion but also extract insights from the visual patterns. These are the core reasons for data visualization. With Parallel Coordinates (abbr. ||-coords) the search for multivariate relations in high dimensional datasets is transformed into a 2-D pattern recognition problem. A geometric classification algorithm based on ||-coords is presented and applied to a complex dataset. It has low computational complexity providing the classification rule explicitly and *visually*. The minimal set of variables required to state the rule is found and ordered *optimally* by their predictive value. A visual economic model of a real country is constructed and analyzed to illustrate how multivariate relations can be modeled by means of hypersurfaces. Recent results like viewing *convexity in any dimension* and non-orientability (as in the Möbius strip) provide a prelude of what is on the way.



Parallel Coordinates. Figure 1. The polygonal line $\bar{C}$ represents the N-tuple $C = (c_1, c_2, c_3, c_4, c_5)$.

Historical Background

Legend has it that while constructing a proof, Archimedes was absorbed in a diagram when he was killed by a Roman soldier. "Do not disturb my circles" he pleaded as he was being struck by the sword. Visualization flourished in Geometry and Archimedes' is the first recorded death in defense of visualization. Visual *interaction* with diagrams is interwoven with testing of conjectures and construction of proofs. The tremendous human *pattern recognition* enables interaction for the extraction of insight from images. This essence of visualization is abstracted and adapted into the general problem-solving process to the extent that, a *mental image* of a problem is formed and at times one says *see* when it is meant *understand*.

Is there a way to make accurate pictures of multidimensional problems analogous to Descartes coordinate system? What is "sacred" about *orthogonal* axes which use up the plane very fast? After all, in Geometry parallelism rather than orthogonality is the fundamental concept and they are not equivalent for orthogonality requires a prior concept of "angle." By 1959, while studying Mathematics at the University of Illinois, these thoughts lead to the *multidimensional* coordinate system based on *Parallel Coordinates*. With the encouragement of Professors Cairns and Bourgin (both topologists) basic properties like the *point* ↔ *line* duality were derived. It was not until 1977 when, while teaching a Linear Algebra course, the "challenge" was raised to *show* spaces of dimension higher than 3; parallel coordinates were recalled and their systematic development began. There was early interest and acceptance [6]. The comprehensive report [7] and later [9] layed the foundations of what became the ||-coords methodology. Collaboration with Dimsdale, (A long-time associate of John von Neuman.) Hurwitz, Boz and Addison ushered to five USA patents (Collision Avoidance Algorithms for Air-Traffic Control, Data Mining and Computer Vision) and applications to Optimization, Process Control [4] and elsewhere. Hurwitz, Adams and later Chomut experimented

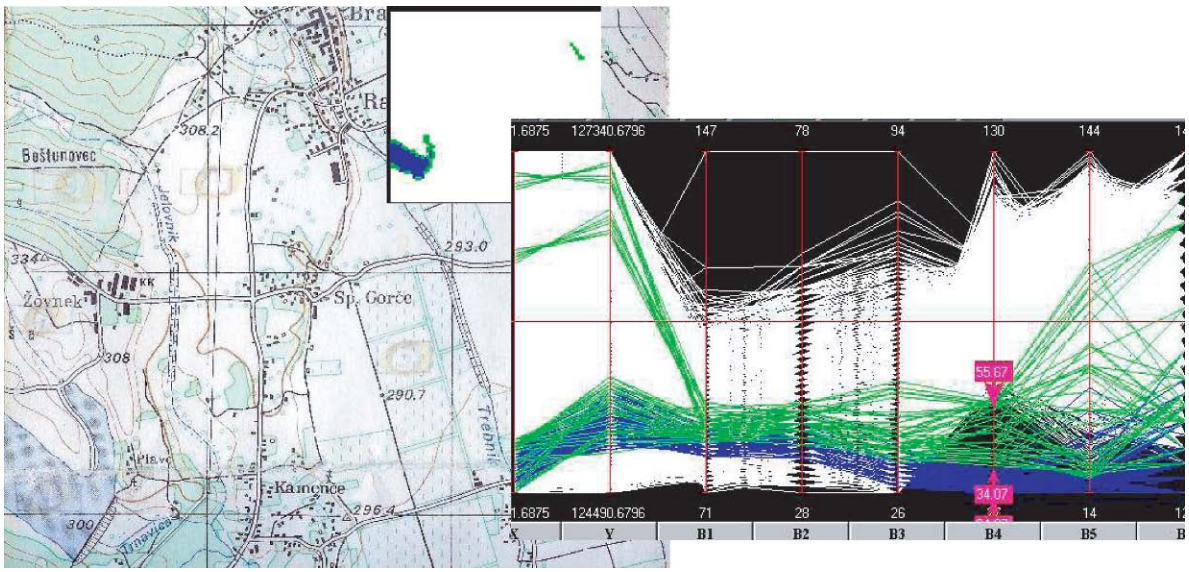
with multi-query interactive software written in APL for exploratory data analysis (EDA) with ||-coords [5]. There followed [10] on the discovery of structure in data, Gennings et al. [4] on response surfaces based on ||-coords for statistical applications, Wegman using the aforementioned duality promoted the EDA application, Fiorini on Robotics and Hinterberger [14] on comparative multivariate visualization (and earlier on data density analysis). The results of Eickemeyer [2], Hung and Inselberg [5] and Chatterjee were seminal. More recently the work of Ward et al. [15] (Hierachical ||-coords and more), Jones [13] (Optimization), Yang (Association Rules – Databases), Hauser (Categorical Variables and more) and Choi and Heejo Lee (on Intrusion Detection), increased the versatility and sophistication of ||-coords. This list is by no means exhaustive, Shneiderman, Grinstein, Keim, Mihalisin, and others have made significant contributions to data and information visualization.

Foundations

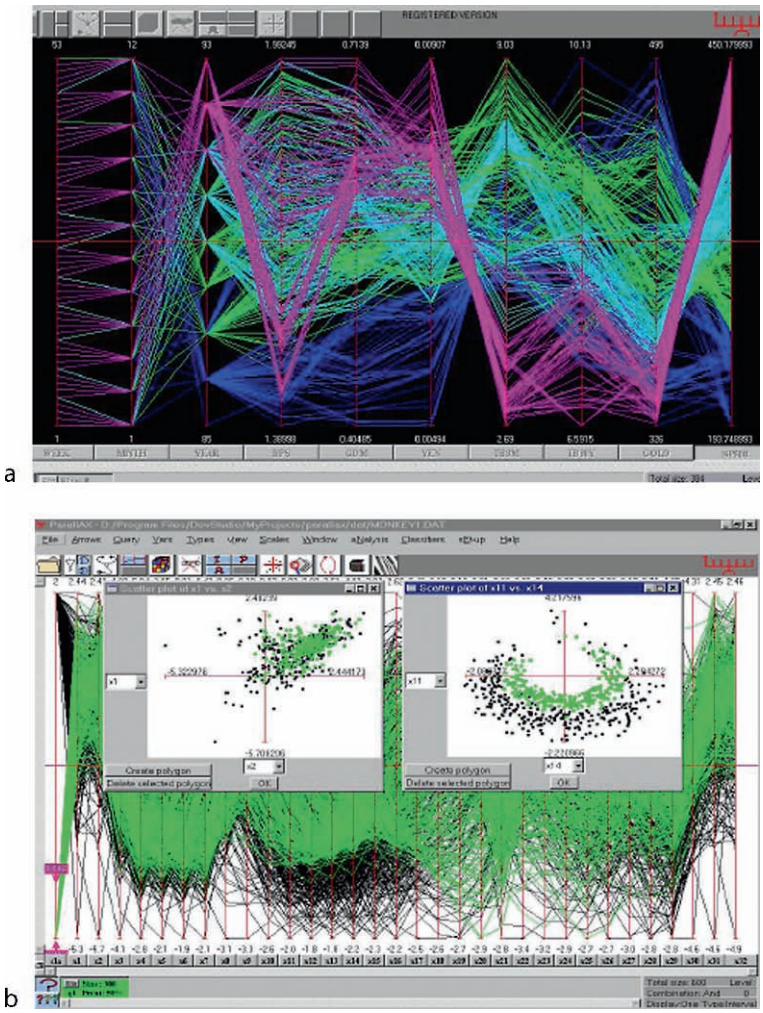
"A picture is worth a thousand words" – But how does one say this with a picture?

The value of data visualization is not seeing "zillions" of objects but rather recognizing *relations* among them. Wonderful successes like Minard's "Napoleon's March to Moscow," Snow's "dot map" and others (see [3]) are *ad hoc* (i.e., one-of-a-kind) and exceptional. Succinct multivariate relations are rarely apparent from *static* displays. With *interactivity* the visual clues in a good data display (*Parallax* MDG's proprietary Data Mining software is used by permission.) can masterfully guide knowledge discovery. Follow up on anything that catches the eyes, gaps, regularities, holes, twists, peaks and valleys, density contrasts like the ones which reveal the water regions in Fig. 2. For the financial data in Fig. 3 (left) *multidimensional contouring* is applied to the axis with *SP500* index values (right) uncovering multivariate relations like high *SP500* low *Gold* and *Interests* correlate with high *Yen* and more. These are examples of two queries and they are others. With Boolean operators compound queries are formed to perform complex tasks. Classification is an important operation in data mining. Powerful geometrical classifiers based on ||-coords [12] have been constructed. An example is shown Fig. 3 (right). The mixing of the two categories is seen on the left plot of the first two parameters. The classifier found the 9 (out of 32) parameters needed to state the rule with 4% error and ordered them according to their predictive value. The two best predictors are plotted on the right showing the separation achieved [12].

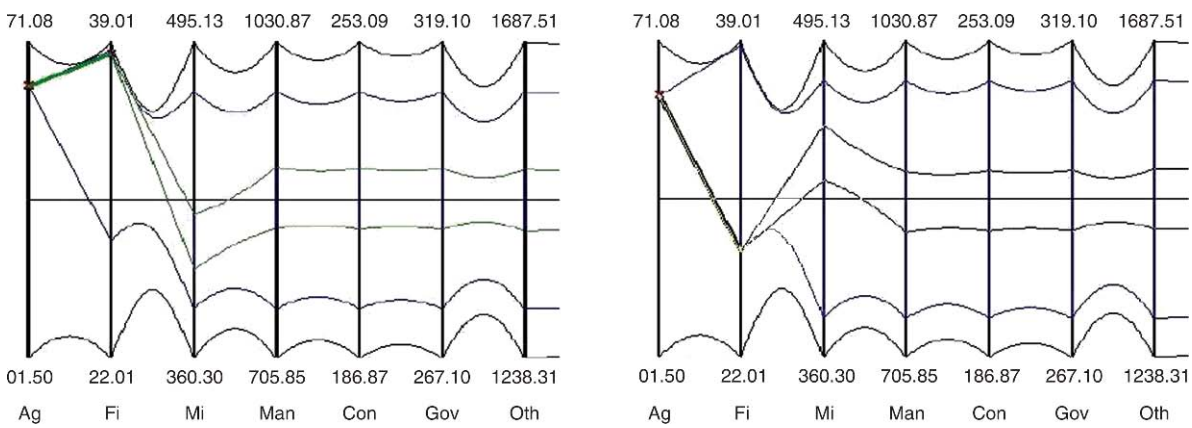
Multivariate relations can be modeled in terms of hypersurfaces – just as a relation between two variables can be represented by a planar region. From a dataset



Parallel Coordinates. Figure 2. Ground emissions measured by satellite on a region of Slovenia (*left*) are displayed on the right. The water and lake's edge are discovered with two queries.

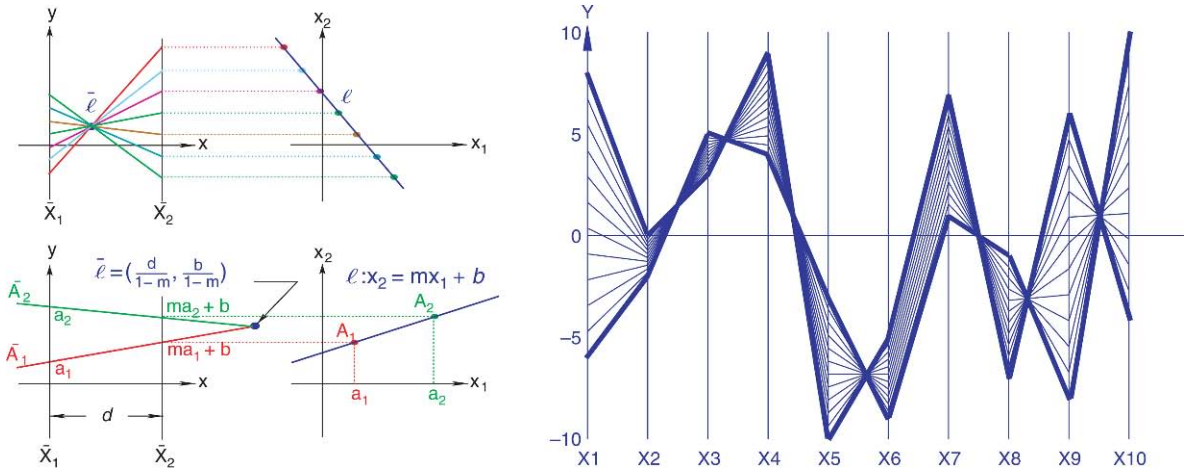


Parallel Coordinates. Figure 3. (a) The *multidimensional-contouring* on financial data reveals multiple interelations. (b) Classification – dataset with 32 parameters and two categories.

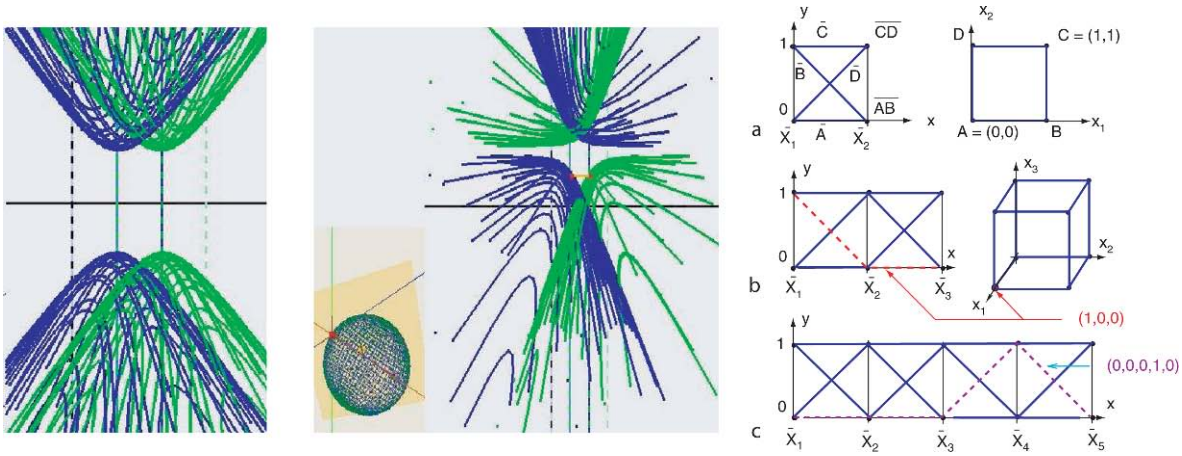


Parallel Coordinates. Figure 4. Hypersurface modeling a country's economy. An interior point represented by a polygonal line depicts a feasible economic policy.

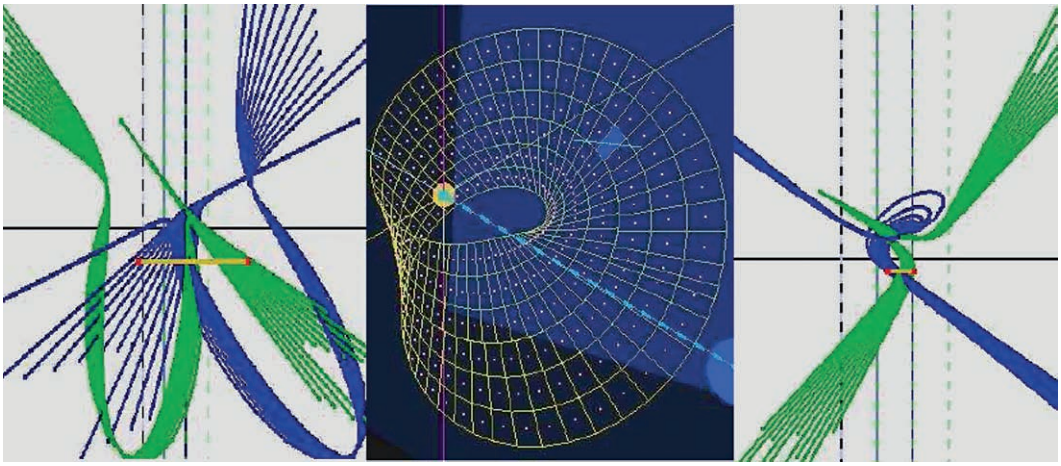
consisting of the outputs of various economic sectors of a real country a visual model of its economy is constructed and shown in Fig. 4 represented by the upper and lower curves. An interior (i.e., a combination of sector outputs) point satisfies all the constraints simultaneously therefore represents a feasible economic policy for that country. Such points can constructed by sequentially choosing variable values within their allowable range. Once a value of the first variable is chosen (in this case the *Agri*cultural output) within its range, the dimensionality of the region is reduced by one. In fact, the upper and lower curves between the second and third axes show the reduced available range of the second variable *Fish*ing and similarly for the remaining variables. Hence the impact of a decision can be seen downstream. In this way it was found that a high value from the available range of *Fish*ing corresponds to very low values of the *Min*ing sector – seen in the right of Fig. 4 and vice versa. This inverse correlation was investigated and found that the two sectors compete for the *same* group of migrating workers. When fishing is doing well, most of them leave the mountains where the mines are located to work on the fishing boats and the reverse. This is an example of ||-coords used for Decision Support and Trade-Off Analysis.



Parallel Coordinates. Figure 5. (*Left*) *point* ↔ *line* duality in 2-D. (*Right*) A N-dimensional line ℓ is represented by $N-1$ points – the intersections of polygonal representing points of ℓ.



Parallel Coordinates. Figure 6. Convex surfaces represented by $N-1$ hyperbola-like regions. (left)Sphere in 3-D centered at origin. (*center*) Tranlated sphere *translation* ↔ *rotation* duality (*right*) cube and hypercube in 5-D repeating pattern.



Parallel Coordinates. Figure 7. Möbius strip and its representation for two orientations.

Key Applications

There are numerous others like GIS, Process Control, Trading & Financial Analysis the visualization and analysis of multivariate/multidimensional problems in general.

Future Directions

Though some refer to ||-coords as a "plot" it is actually *a coordinate system* where the goal, since its inception, is to concentrate N-dimensional relational information into *patterns* the earliest being that N-dimensional lines are represented by $N-1$ points with *two indices* Fig. 5. Hyperplanes are represented by $N-1$ points with N indices, and $N-1$ regions with N indices represent a surface – i.e., the tangent planes enveloping it. Figures 6 and 7 show recent breakthroughs where *convexity in any dimension* and *non-orientability* can be seen. Work is progressing on transforming multivariate relations in datasets into planar patterns completely eliminating display clutter [11].

Cross-references

- ▶ Abstraction
- ▶ Association Rule Mining on Streams
- ▶ Business Intelligence
- ▶ Classification
- ▶ Clustering
- ▶ Cluster Visualization
- ▶ Comparative Visualization
- ▶ Curse of Dimensionality
- ▶ Data Mining
- ▶ Data Visualization
- ▶ Data Warehouse
- ▶ Dimension
- ▶ Dimension Reduction Techniques for Clustering
- ▶ Dimensionality Reduction
- ▶ Dynamic Graphics
- ▶ Event Detection
- ▶ Exploratory Data Analysis
- ▶ Feature Selection for Clustering
- ▶ Geographic Information System
- ▶ High Dimensional Indexing
- ▶ Individually Identifiable Data
- ▶ Information Extraction
- ▶ Interface
- ▶ Machine Learning in Computational Biology
- ▶ Mining of Chemical Data
- ▶ Model Management
- ▶ Multidimensional Data Formats
- ▶ Multidimensional Modeling
- ▶ Multidimensional Scaling
- ▶ Multivariate Visualization Methods
- ▶ OLAP
- ▶ Parallel Visualization
- ▶ Principal Component Analysis
- ▶ Process Optimization
- ▶ Query Evaluation Techniques for Multidimensional Data
- ▶ Range Query
- ▶ Scientific Visualization
- ▶ Spatial Data Analysis
- ▶ Spatial Data Mining
- ▶ Spatio-Temporal Data Mining
- ▶ Temporal Data Mining
- ▶ Temporal Dependencies

- ▶ Visual Analytics
- ▶ Visual Association Rules
- ▶ Visual Classification
- ▶ Visual Clustering
- ▶ Visual Interfaces
- ▶ Visualization Techniques for Scientific Databases
- ▶ Visualizing Quantitative Data
- ▶ What-if Analysis

Recommended Reading

1. Chomut T. Exploratory data analysis in parallel coordinates. M. Sc. Thesis, Department of Computer Science, UCLA, 1987.
2. Eickemeyer J. Visualizing p-Flats in N-Space Using Parallel Coordinates. Ph.D. Thesis, Department of Computer Science, UCLA, 1992.
3. Friendly M. et al. Milestones in Thematic Cartography. www.math.yorku.ca/scs/SCS/Gallery/milestones/, 2005.
4. Gennings C., Dawson K.S., Carter W.H., and Myers R.H. Interpreting plots of a multidimensional dose-response surface in parallel coordinates. Biometrics, 46:719–35, 1990.
5. Hung C.K. and Inselberg A. Parallel Coordinate Representation of Smooth Hypersurfaces. USC Tech. Report # CS-92-531, Los Angeles, 1992.
6. Inselberg A. N-dimensional coordinates. In Proc. of IEEE Conf. Picture Data Description, 1980.
7. Inselberg A. N-dimensional graphics, LASC Tech. Rep. G320-2711, IBM, 1981.
8. Inselberg A. Intelligent instrumentation and process control. In Proc. Second IEEE Conf. on AI Application, 1985, pp. 302–307.
9. Inselberg A. The plane with parallel coordinates. Visual Computer, 1:69–97, 1985.
10. Inselberg A. Discovering multi-dimensional structure with parallel coordinates (*invited paper*). In Proc. ASA.– Stat. Graphics *1*–16, 1989.
11. Inselberg A. Parallel Coordinates : *VISUAL* Multidimensional Geometry and its Applications. Springer, 2009.
12. Inselberg A. and Avidan T. The *Automated* Multidimensional Detective. In Proc. IEEE Information Visualization, 1999, pp. 112–119.
13. Jones C. Visualization and optimization. Kluwer Academic, Boston, 1996.
14. Schmid C. and Hinterberger H. Comparative Multivariate Visualization Across Conceptually Different Graphic Displays, In Proc. 6th Int. Conf. on Scientific and Statistical Database Management, 1994.
15. Ward M.O. XmdvTool: integrating multiple methods for visualizing multivariate data. In Proc. IEEE Conf. on Visualization, 1994, pp. 326–333.

Parallel Coordinates Plot (PCP)

- ▶ Parallel Coordinates

Parallel Coordinates System (PCS)

- ▶ Parallel Coordinates

Parallel Data Placement

Patrick Valduriez
INRIA, LINA, Nantes, Cedex, France

Synonyms

Multiprocessor data placement

Definition

Parallel data placement refers to the physical placement of the data in a multiprocessor computer in order to favor parallel data access and yield high-performance. Most of the work on data placement has been done in the context of the shared-nothing architecture. Data placement in a parallel database system exhibits similarities with data fragmentation in distributed databases since fragmentation yields parallelism. However, there is no need to maximize local processing (at each node) since users are not associated with particular nodes and load balancing is much more difficult to achieve in the presence of a large number of nodes.

The main solution for parallel data placement is a variation of horizontal fragmentation, called *partitioning*, which divides database relations into partitions, each stored at a different disk node. There are three basic partitioning strategies: round-robin, hashing, and interval. Furthermore, to improve availability, replication of partitions is needed.

Historical Background

Parallel data placement was proposed in the early 1980s for the first software-oriented parallel database systems. Instead of proposing expensive changes to the disk technology as in database machines with filtering devices, the main idea was to distribute the data onto multiple (smaller) disks and parallelize the I/O bandwidth which could be better consumed by multiple processors.

Data partitioning was first proposed for shared-nothing architectures which are a natural evolution of distributed database architectures. The same basic idea has been also used (with different partitioning

strategies) for the Redundant Arrays of Inexpensive Disks (RAID) to build powerful disks out of many smaller ones.

Most of the work in parallel data placement has been done in the context of the relational model with relation partitioning as the main strategy. Extensions have been proposed for object-oriented or semi-structured (XML) data by partitioning collections of objects or elements.

Foundations

Most of the work on data placement has been done in the context of the shared-nothing architecture which is the most general architecture. However, data placement is also important in shared-memory and shared-disk architectures since there can be multiple disks. Thus, the data placement techniques designed for shared-nothing can also be used, sometimes in a simplified form, to other architectures.

Data placement in a parallel database system exhibits similarities with data fragmentation in distributed databases. An obvious similarity is that fragmentation can be used to increase parallelism. In what follows, the terms partitioning and partition are used instead of horizontal fragmentation and horizontal fragment, respectively, to contrast with the alternative approach which clusters all data of each relation at one node. Vertical fragmentation can also be used to increase parallelism and load balancing much as in distributed databases. Another similarity is that queries should be executed as much as possible where the data reside. However, there are two important differences with the distributed database approach. First, there is no need to maximize local processing (at each node) since users are not associated with particular nodes. Second, load balancing is much more difficult to achieve in the presence of a large number of nodes. The main problem is to avoid resource contention, which may result in the entire system thrashing (e.g., one node ends up doing all the work while the others remain idle). Since queries are executed where the data reside, data placement is a critical performance issue.

Data placement must be done to maximize system performance, which can be measured by combining the total amount of work done by the system and the response time of individual queries. Maximizing response time (through intra-query parallelism) may result in increased total work due to communication overhead. For the same reason, inter-query parallelism results in increased total work. On the other hand, clustering all the data necessary to a query minimizes communication and thus the total work done by the system in executing that query. In terms of data placement, the following trade-off exists: maximizing response time or inter-query parallelism leads to partitioning, whereas minimizing the total amount of work leads to clustering. This problem is addressed in distributed databases in a rather static manner. The database administrator is in charge of periodically examining fragment access frequencies, and when necessary, moving and reorganizing fragments.

A solution to data placement is full partitioning, whereby each relation is horizontally fragmented across all the nodes in the system. There are three basic strategies for data partitioning: round-robin, hash, and range partitioning.

1. *Round-robin partitioning* is the simplest strategy, it ensures uniform data distribution. With n partitions, the ith tuple in insertion order is assigned to partition (i mod n). This strategy enables the sequential access to a relation to be done in parallel. However, the direct access to individual tuples, based on a predicate, requires accessing the entire relation.
2. *Hash partitioning* applies a hash function to some attribute which yields the partition number. This strategy allows exact-match queries on the selection attribute to be processed by exactly one node and all other queries to be processed by all the nodes in parallel.
3. *Range partitioning* distributes tuples based on the value intervals (ranges) of some attribute. In addition to supporting exact-match queries as with hashing, it is well-suited for range queries. For instance, a query with a predicate "A between a_1 and a_2 may be processed by the only node(s) containing tuples whose A value is in the range [a_1 and a_2]." However, range partitioning can result in high variation in partition size.

Full partitioning generally yields better performance than clustering relations on a single (possibly very large) disk. Although full partitioning has obvious performance advantages, highly parallel execution may cause a serious performance overhead for complex queries involving joins. Furthermore, full partitioning is not appropriate for small relations that span a few disk blocks. These drawbacks suggest that a compromise between clustering and full partitioning needs to be found.

A solution is to do data placement by variable partitioning, where the degree of partitioning, in other words, the number of nodes over which a relation is partitioned, is a function of the size and access frequency of the relation. This strategy is much more involved than either clustering or full partitioning because changes in data distribution may result in reorganization. For example, a relation initially placed across eight nodes may have its cardinality doubled by subsequent insertions, in which case it should be placed across 16 nodes.

In a highly parallel system with variable partitioning, periodic reorganizations for load balancing are essential and should be frequent unless the workload is fairly static and experiences only a few updates. Such reorganizations should remain transparent to compiled queries that run on the database server. In particular, queries should not be recompiled because of reorganization. Therefore, the compiled queries should remain independent of data location, which may change rapidly. Such independence can be achieved if the run-time system supports associative access to distributed data. This is different from a distributed DBMS, where associative access is achieved at compile time by the query processor using the data directory.

A serious problem in data placement is dealing with skewed data distributions which may lead to non-uniform partitioning and hurt load balancing. Range partitioning is more sensitive to skew than either round-robin or hash partitioning. A solution is to treat non-uniform partitions appropriately, e.g., by further fragmenting large partitions.

Another important function is data replication for high availability. The simple solution is to maintain two copies of the same data, a primary and a backup copy, on two separate nodes. This is the mirrored disks architecture promoted by many computer manufacturers. However, in case of a node failure, the load of the node having the copy may double, thereby hurting load balancing. To avoid this problem, an interesting solution is Teradata's interleaved partitioning which partitions the backup copy on a number of nodes. In failure mode, the load of the primary copy gets balanced among the backup copy nodes. But if two nodes fail, then the relation cannot be accessed thereby hurting availability. Reconstructing the primary copy from its separate backup copies may be costly. In normal mode, maintaining copy consistency may also be costly. A solution to this problem is Gamma's chained partitioning which stores the primary and backup copy on two adjacent nodes. The main idea is that the probability that two adjacent nodes fail is much less than the probability that any two nodes fail. In failure mode, the load of the failed node and the backup nodes are balanced among all remaining nodes by using both primary and backup copy nodes. In addition, maintaining copy consistency is cheaper.

Key Applications

Parallel data placement has been primarily used by parallel database systems on multiprocessor computers. Data partitioning is also used as a basic technique for dealing with very large volumes of data in different environments such as database clusters, data grids, search engines, document management systems, and P2P systems.

Cross-references

▶ Distributed Databases

Recommended Reading

1. Copeland G.P., Alexander W., Boughter E.E., and Keller T.W. Data placement in bubba. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988, pp. 99–108.
2. DeWitt D.J. and Gray J. Parallel database systems: the future of high performance database systems. Commun. ACM, 35(6):85–98, 1992.
3. Mehta M. and DeWitt D.J. Data placement in shared-nothing parallel database systems. VLDB J., 6(1):53–72, 1997.
4. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 3rd edn. Prentice Hall, 2009.
5. Valduriez P. and Pacitti E. Parallel database systems. In Handbook of Database Technology, J. Hammer, M. Scheider (eds.). CRC Press, Boca Raton, FL, 2007.

Parallel Database

▶ Distributed Architecture

Parallel Database Management

Patrick Valduriez
INRIA, LINA, Nantes, Cedex 3, France

Synonyms

Multiprocessor database management

Definition

Parallel database management refers to the management of data in a multiprocessor computer and is done by a parallel database system, i.e., a full-fledge DBMS implemented on a multiprocessor computer. The basic principle employed by parallel DBMS is to partition the data across multiprocessor nodes, in order to increase performance through parallelism and availability through replication. This enables supporting very large databases with very high query or transaction loads.

Parallel database systems can exploit distributed database techniques. In particular, database partitioning is somewhat similar to database fragmentation. Essentially, the solutions for transaction management, i.e., distributed concurrency control, reliability, atomicity, and replication, can be reused. However, the critical issues for parallel database systems are data placement, parallel query processing (including optimization and execution) and load balancing. These issues are much more difficult than in distributed DBMS because the number of nodes may be much higher. Furthermore, the interconnection network of a multiprocessor system provides better opportunities for improving performance than a general-purpose network.

Historical Background

The performance objective of parallel database systems was also that of the database machines in the 1980s. The problem faced by conventional database management on a monoprocessor computer has long been known as "I/O bottleneck," induced by high disk access time. Database machine designers tackled this problem through special-purpose hardware, e.g., by introducing data filtering devices within the disk heads. However, this approach did not succeed because of a poor cost/performance ratio compared to the software solution which can easily benefit from progress in standard hardware technology (processor, RAM, disk). However, the idea of pushing database functions closer to the disk has received renewed interest with the introduction of general-purpose microprocessors in disk controllers, thus leading to intelligent disks.

An important result of database machine research, however, is in the general solution to the I/O bottleneck which can be summarized as increasing the I/O bandwidth through parallelism. If a database is partitioned, e.g., by horizontal fragmentation of the relational tables, across a number d of disks, then disk throughput can be increased by accessing the d disks in parallel. The same principle has been applied to Redundant Arrays of Inexpensive Disks (RAID) which build a powerful disk with many small disks.

The main memory database solution which maintains the entire database in main memory to avoid disk accesses, is complementary rather than alternative to parallel database systems. In particular, the "memory access bottleneck" induced by high memory access time relative to processor speed can also be tackled using parallel database techniques. In other words, a parallel database system can use main memory database nodes.

Foundations

Parallel database system designers strive to develop software-oriented solutions in order to exploit the various kinds of multiprocessor systems which are now available. This can be achieved by extending distributed database technology, for example, by partitioning the database across multiple disks so that much inter- and intra-query parallelism can be obtained. This can lead to significant improvements in both response time and throughput (number of transactions per time unit).

A parallel database system can be loosely defined as a DBMS implemented on a multiprocessor computer. This definition includes many alternatives ranging from the porting of an existing DBMS, which may require only rewriting the operating system interface routines, to a sophisticated combination of parallel processing and database system functions into a new hardware/software architecture. Thus, there is a trade-off between portability (to several platforms) and efficiency. The sophisticated approach is better able to fully exploit the opportunities offered by a multiprocessor at the expense of portability. Interestingly, this gives different advantages to computer manufacturers and software vendors.

The main objectives of a parallel database system are the following:

High-performance. This can be obtained through several complementary solutions: data-based parallelism, query optimization, and load balancing. Parallelism can increase throughput, using inter-query parallelism, and decrease transaction response times, using intra-query parallelism. Load balancing is the ability of the system to divide a given workload equally among all processors.

Depending on the parallel system architecture, it can be achieved by static physical database design or dynamically at run-time.

High-availability. Because a parallel database system consists of many redundant components, it can well increase data availability and fault-tolerance. Replicating data at several nodes is useful to support failover, a fault-tolerance technique which enables automatic redirection of requests from a failed node to another node which stores a copy of the data, thereby providing nonstop operation.

Extensibility. In a parallel system, accommodating increasing database sizes or increasing performance demands (e.g., throughput) should be easier. Extensibility is the ability of smooth expansion of the system by adding processing and storage power to the system. Ideally, the increase in the configuration of the parallel database system (i.e., more nodes) should speed up existing workloads or scale up to larger databases.

The functions supported by a parallel database system are the same as in a typical DBMS. The differences, though, have to do with implementation of these functions which must now deal with parallelism, data partitioning and replication, and distributed transactions. Depending on the architecture, a processor node can support all (or a subset) of these functions. In the early database machines for instance, some nodes were specialized with hardware to implement some functions, e.g., data filtering.

The way the main hardware elements, i.e., processors, main memory, and disks, are connected (through some interconnection network) has a major impact on the design of a parallel database system. The term processor is used in the general sense of central processing unit (CPU). However, the processor itself can be made of several core processors integrated in a single chip, i.e., a multi-core processor, and perform instruction-level multithread parallelism. But the parallelism being discussed here is much higher-level (query- or operator-level) so processors (e.g., multi-core processors) are simply considered as black-box components accessing other resources (main memory, disk, network). Depending on how main memory or disk is shared, three basic architectures are obtained: shared-memory, shared-disk and shared-nothing.

In the shared-memory architecture, any processor has access to the entire main memory, and thus to all the disks, through the interconnection network. The network is typically very fast (e.g., a high-speed bus or a cross-bar switch) and redundant to avoid any unavailability. All the processors are under the control of a single operating system. Shared-memory makes it easy to build a parallel database system because the operating system deals with load balancing. It is also very efficient since processors can communicate via the main memory. However, the complexity and cost of the interconnection network hurt extensibility which is limited to a few tens of processors. Most parallel database system vendors provide support for shared-memory.

In the shared-disk architecture, any processor has access to any disk unit through the interconnection network but exclusive (non-shared) access to its main memory. Each processor-memory node is under the control of its own copy of the operating system. An important function that is needed is cache coherency which allows different nodes to cache a consistent disk page. This function is hard to support and requires some form of distributed lock management. Shared-disk has better extensibility than shared-memory since the main memory is distributed. However, scalability depends on the efficiency of the cache coherency mechanism. Migrating from a centralized system to shared-disk is relatively straightforward since the data on disk need not be reorganized. Shared-disk is the main architecture used by Oracle for its parallel database system.

The shared-nothing architecture is fully-distributed: each node is made of processor, main memory and disk and communicates with other nodes through the interconnection network.

Similar to shared-disk, each processor-memory-disk node is under the control of its own copy of the operating system. Then, each node can be viewed as a local site (with its own database and software) in a distributed database system. Therefore, most solutions designed for distributed databases such as database fragmentation (called partitioning in parallel databases), distributed transaction management and distributed query processing may be reused. Using a fast interconnection network, it is possible to accommodate large numbers of nodes. The major advantages of shared-nothing over shared-memory or shared-disk are those of distributed systems: relatively low cost, extensibility and availability. However, it is much more complex to manage as physical data portioning is necessary. Shared-nothing has been adopted by the major DBMS vendors (except Oracle) for their high-end parallel database systems.

Various possible combinations of these three basic architectures are possible to obtain different trade-offs between cost, performance, extensibility, availability, etc. Hierarchical architectures typically combine the three architectures using a shared-nothing design where each node can be shared-memory or shared-disk and communicates with other nodes through the interconnection network.

Key Applications

Parallel database systems are used to support very large databases (e.g., tens or hundreds of terabytes). Examples of applications which deal with very large databases are e-commerce, data warehousing, and data mining. Very large databases are typically accessed through high numbers of concurrent transactions (e.g., performing on-line orders on an electronic store) or complex queries (e.g., decision-support queries). The first kind of access is representative of On Line Transaction Processing (OLTP) applications while the second is representative of On Line Analytical Processing (OLAP) applications. Although both OLTP and OLAP can be supported by the same parallel database system (on the same multiprocessor), they are typically separated and supported by different systems to avoid any interference and ease database operation.

Cross-references

► Distributed Databases

Recommended Reading

1. DeWitt D.J. and Gray J. Parallel database systems: the future of high performance database systems. Commun. ACM, 35 (6):85–98, 1992.
2. Özsu T. and Valduriez P. Distributed and parallel database systems – Technology and current state-of-the-art. ACM Comput. Surv., 28(1):125–128, 1996.
3. Özsu M.T. and Valduriez P. Principles of Distributed Database Systems, 3rd edition. Prentice Hall, 2009.
4. Valduriez P. Parallel database systems: the case for shared-something. In Proc. 9th Int. Conf. on Data Engineering, 1993, pp. 460–465.
5. Valduriez P. and Pacitti E. Parallel Database Systems. In Handbook of Database Technology, J. Hammer, M. Scheider (eds.). CRC Press, Boca Raton, FL, 2007.

Parallel Distributed Processing

► Neural Networks

Parallel Hash Join, Parallel Merge Join, Parallel Nested Loops Join

Goetz Graefe
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

Parallel join algorithms

Definition

These join algorithms are parallel versions of the traditional serial join algorithms. They are designed to exploit multiple processors on a network, within a machine, or even within a single chip.

Key Points

Parallel join algorithms are based on the traditional serial join algorithms, namely (index) nested loops join, merge join, and (hybrid) hash join. The goal of parallel execution is to reduce the input sizes in each processing element, thus reducing the time for query completion even at the expense of increasing overall query execution effort due to data movement. Ideally, parallel join algorithms exhibit linear speed-up and linear scale-up.

Parallel join algorithms are orthogonal to pipelining among join operations in a complex query execution plan. Even a query with a single join can benefit from a parallel join algorithm. The essence of pipelining (and also of "bushy parallelism" in appropriate query execution plans) is to execute different operations with different predicates concurrently, whereas the essence of parallel join algorithms is that a single logical operation exploits multiple processors or processing cores. The conceptual foundations of parallel join algorithms are that database query processing manipulates sets of records, that sets can be divided into disjoint subsets, and that query results based on one record can be computed independently of results based on other records.

Parallel join algorithms apply not only to inner joins but also to semijoins, outer joins, and set operations such as intersection, union, and difference. Thus, they apply to query execution techniques such as index intersection for conjunctive predicates and the star join techniques that are generalizations traditional binary joins.

For a parallel join algorithm, input records are partitioned or replicated such that each result record

is produced exactly once. The usual choices are to partition both inputs or to partition one input and to replicate the other. For parallel joins without any equality predicate, the processing elements can be organized in a rectangular grid, one input partitioned over rows of the grid and replicated within each row, and the other join input partitioned over columns of the grid and replicated within each column. These choices mirror schemes in which database records may be stored partitioned or replicated.

Partitioning schemes include range partitioning, hash partitioning, and hybrid schemes such as range partitioning of hash values or hashing (identifiers of) key ranges. For intermediate query results, hash partitioning is simple yet reasonably robust against skewed key distributions.

All of the join algorithms can reduce data transfer efforts using semijoin reduction or its heuristic approximation, bit vector filtering or Bloom filters. Hash join can benefit most obviously due to its separate build and probe phases, with the bit vector filter populated during the build phase and exploited during the probe phase. Merge join can exploit bit vector filtering if the filter can be populated prior to a stop-and-go operation such as sort, and can exploit it with the most gain if the other input also must be processed by an expensive operation such as a sort.

Symmetric partitioning means that both inputs are partitioned. Ideally, the inputs are stored partitioned in such a way that join processing does not incur any cost or delay for partitioning. This is often called "local indexing" or "table partitioning" when it refers to indexes of the same table and "aligned partitioning" when it refers to separate tables. Aligned partitioning often coincides with frequent join operations, foreign key constraints, and complex objects on the conceptual level. The opposites are "global indexing," "index partitioning," and "non-aligned partitioning."

If one of the join inputs is very large, and if the degree of parallelism is only moderate, it might be less expensive to replicate the small input than to partition the large input. An appropriate cost calculation needs to consider record counts, record sizes, communication costs per message or per byte, the available memory allocation for each thread, etc. A first approximation compares the quotient of the two input sizes with the degree of parallelism.

Cost calculation during compile-time query optimization can focus either on the average cost in each of the parallel processing elements or on the maximal cost. The former metric focuses on overall system throughput in multi-user systems, whereas the latter metric focuses on the query elapsed time as perceived by a single user or application.

Cross-references

▶ Parallel Query Execution Algorithms

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.
2. Mishra P. and Eich M.H. Join processing in relational databases. ACM Comput. Surv., 24(1):63–113, 1992.

Parallel Join Algorithms

▶ Parallel Hash Join, Parallel Merge Join, Parallel Nested Loops Join

Parallel Query Execution Algorithms

Goetz Graefe
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

Partitioned query execution; Partitioning

Definition

Parallel query execution algorithms employ multiple threads that together execute a single query execution plan, with the goal of increased processing bandwidth and decreased response time. Multiple forms of parallelism may be employed in isolation or in combination. Most parallel query execution algorithms are serial algorithms executed in multiple threads, usually with no or only minor adaptations to function correctly or efficiently in the multi-threaded environment.

Historical Background

Set-oriented query languages and the relational algebra have given rise to query optimization and parallel database processing almost from their beginning in the 1970s. They represented a new complexity or

liability in database software design as non-procedural languages and physical data independence prevent users from specifying query processing steps and thus require an automatic system component to provide plans for data access and for query execution. At the same time, they created a new opportunity as to prior database systems and their interfaces did permit query rewrite techniques or parallel execution.

The last 30 years have seen many successes and many failures in parallel query execution algorithms. Parallel versions of naïve nested loops join were a failure, because even hardware acceleration cannot compete with well-chosen indexes, whether those are persistent indexes supporting index nested loops join or temporary indexes in form hash tables in hash join. Parallel set operations, including non-indexed selection, join, grouping, and sorting, have been great successes, because they can provide both good speed-up and scale-up. Parallelism is sometimes associated with specific set operations, e.g., hash join instead of all joins, but almost all join algorithms benefit quite similarly if designed and implemented with similar care.

Foundations

In principle, parallel query execution algorithms are traditional sequential algorithms with multiple threads executing different operations or executing the same operation on different data. The relational algebra of sets permits pipelining between operations, partitioning of stored data and of intermediate results, and bushy execution in complex query execution plans. Individual parallel algorithms exploit partitioning but also overcome the issues introduced by partitioning and parallelism.

The value and success of parallel query execution is often expressed as speed-up and scale-up. Speed-up compares execution times of serial and parallel execution for a constant data volume. Linear speed-up is considered ideal; it is achieved if the ratio of execution times is equal to the degree of parallelism. Scale-up compares execution times for a constant ratio of data volume and degree of parallelism. Linear scale-up is considered ideal; it is achieved if the execution times remain constant. There are, however, some effects that permit super-linear speed-up. For example, adding nodes to a parallel database machine increases not only processing bandwidth but also available memory, and might thus turn an external sort into an in-memory sort at a high degree of parallelism.

Pipelining

The pipeline between a producer operation and a consumer operation may pass individual records, value packets, pages, or even runs. Value packets within a sorted stream are defined by equal values in the sort columns. As an example for iterators passing run information, consider a sort operation split into run generation and merging; the pipeline between those two components may pass entire runs at-a-time. In addition to data, the pipeline contents may include control records, e.g., artificially created keys design to advance the merge logic further up in the sequence of operations.

More importantly for multi-threaded query execution, the pipeline between threads may be demand-driven or data-driven. Within each thread, iterators commonly implement demand-driven dataflow, which lend themselves well to joins and other binary operations but not to sharing intermediate results or common sub-expressions, which occur much less frequently than joins in query execution plans. Between threads, data-driven dataflow is commonly regarded superior.

On the other hand, if flow control is implemented, the slower one among producer and consumer dictates the overall pipeline flow, and it might not truly be useful to attempt distinguishing demand- and data-driven dataflow. The important aspect is that some "slack" is created that permits the producer to get ahead of the consumer to some reasonable amount without requiring an excessive amount of buffer space. In general, the quotient of buffer space and bandwidth is equal to the interval that can be smoothed out, i.e., the amount of time the producer may get ahead of the consumer or the consumer may fall behind the producer.

Partitioning

Since pipelining with flow control often leads to consumers throttling producers or the other way around, and since pipelining alone permits only a fairly limited degree of parallelism, it is usually combined with partitioning. Partitioning means that the input into a relational algebra operation is divided into disjoint subsets, which are processed by separate threads or processors.

Many partitioning methods are in common use. The basic partitioning methods are random, round-robin, range, and hash. Both random and round-robin partitioning permit perfect load balancing but do not

exploit key values and their importance in relational algebra operations such as joins and duplicate elimination. When random partitioning or round-robin partitioning is applied to stored data, neither queries nor updates can be directed to a single location and instead must be processed in all partitions. Range partitioning assigns key ranges (of the partitioning column) to storage or processing sites. For perfect load balancing, accurate quantile information is required. For intermediate results, quantiles must be estimated during query optimization or during an earlier query execution phase. Hash partitioning applies a hash function to determine storage or processing sites. A suitable hash function can achieve very good load balancing, which is why hash partitioning is sometimes associated with linear speed-up and scale-up. On the other hand, even hash partitioning fails if the data distribution suffers from duplicate skew, i.e., the data contain many duplicate key values.

There are many ways to combine these basic partitioning methods. For example, it may be desirable that small range queries are processed at a one or two sites with little overhead and that large range queries are processed by many or all sites with maximum throughput. To achieve both these effects, many small key ranges can be hashed to storage sites. In other cases, one column may be used to hash data items to sites and another column to hash data items to storage locations within each site. This is just one of many multi-dimensional partitioning methods. Note that partitioning among sites, partitioning within sites, and local organization are orthogonal. The possibilities are endless, and almost each vendor claims to have found a new and superior "secret sauce." The most sophisticated systems even offer partitioning on computed columns or expressions.

Partitioning of stored data usually determines where initial query operations execute, including selection and projection as well as preliminary stages of sorting, duplicate elimination, and "group by" operations. Partitioning of stored data may also interact with high availability consideration, e.g., each data item is hashed to two sites using two hash functions, carefully designed to ensure that no data item is hashed to the same site by both hash functions.

For binary operations, "co-location" of the two inputs permits local execution. These binary operations include inner and outer joins, semijoins including "exists" nested queries, and set operations such as intersection, union, and difference. Set operations after scans of non-clustered indexes are common in query execution plans for complex queries, for ad-hoc queries, and for star joins in relational data warehouses using a star schema. If all indexes for a table or view are partitioned in the same way, both set operations and fetch operations can remain local. On the other hand, there are many situations in which it is advantageous to partition each index on its search columns.

If co-location and local execution are not possible, join input data need to be re-partitioned. In most cases, both inputs are partitioned on the columns participating in the join operation's equality predicate. If one input is much smaller than the other, it may be more efficient to broadcast the small input and not move the large input. In some case, in particular non-equality joins, a combination can be applied. In this technique, the processing sites can be thought of as a rectangular matrix, with one input partitioned among rows and broadcast within rows, and the other input partitioned among columns and broadcast within columns. The essence of all partitioning strategies for binary operations such as joins is to ensure that each pair of input rows that may contribute to the join result "meets" at precisely one processing site.

Bushy Execution

In addition to pipelining and partitioning, a third form of parallel query execution is enabled by bushy query execution plans, as opposed to linear query execution plans. In a linear query execution plan, one input of each binary operation always is a stored table. In a bushy query execution plan, both inputs may be intermediate query results. For example, two sort operations may provide the inputs for a merge join. During the sort operations' input phases, the two sort operations can run in parallel. In more complex query execution plans, numerous operations and plan branches may be active concurrently and independently.

Even the simple example with two sort operations and a merge join suffices to demonstrate the substantial difficulties, however. One problem is finding an appropriate degree of parallelism for each operation or branch. A harder problem is assigning appropriate resources such as memory, disk space and bandwidth, etc. The hardest problem is to ensure that concurrent branches produce their results at the right time. In the example, resources may be wasted if one sort operation finishes before the other, because the former will likely

hold on to resources such as memory while waiting for the merge join to need its input and thus for the other sort operation to finish.

Specifics of Parallel Algorithms

In order to keep parallel query execution engines modular, traditional serial query execution algorithms are combined with mechanisms for parallel execution, specifically pipelining and partitioning. The interface between traditional serial query execution operations usually takes the form of iterator. Iterator methods support forward-only scans of intermediate results, rewinding, binding parameters as needed for nested iteration and general predicate evaluation, etc. Iterator instances may recursively invoke their input iterators to implement those methods, and by doing so can execute a very complex query execution plan within a single thread.

Mechanisms for parallel query execution can be encapsulated in a special iterator. This design has been used in a number of research prototypes and commercial database systems. Commonly called the "exchange" operation but also known as "river" or "asynchronous table queue," this iterator encapsulates data transfer and flow control, often provides initialization and teardown of threads and communication paths, and batches records in order to reduce inter-process communication. If parallel execution of nested iteration is desired, both data transfer and flow control must work both from producer to consumer for intermediate query results and from consumer to producer for bindings of correlation columns from outer to inner inputs of nested iteration operations.

As the exchange operation or its equivalent encapsulate the basic mechanisms for parallel query execution, the following only indicates how specific parallel algorithms differ from their serial equivalents.

For parallel sorting, after the input data is partitioned as appropriate, the individual threads of a parallel sort operation usually can work entirely independently. If, however, the query optimizer relies on the sort operation not only for the correct sequence of records but also to avoid the Halloween problem by means of phase separation, the individual threads must coordinate their transition from consuming unsorted input records to producing sorted output records. If one of them produces output too early, it might affect the input set of another thread still consuming input and thus create the Halloween problem.

A similar coordination point exists in parallel hash join. A serial hash join might choose between Grace join, which devotes its first phase entirely to partitioning, and hybrid hash join, which performs some join logic even during its first phase. Multiple threads in a parallel hash join might choose differently, e.g., due to different data volumes. Scheduling other operations within the query execution plan might require, however, that all hash join threads follow the same choice. As for parallel sorting with built-in Halloween protection, such a parallel hash join requires a very brief communication and decision phase.

For parallel hash aggregation and duplicate elimination, a common technique is to run the operation twice, once before and once after data partitioning, in cases in which the input data are not yet partitioned on the grouping columns. The goal is to reduce the number of data records that need to be shipped between threads, processes, and processors. Assuming uniform random distribution of input rows prior to re-partitioning, this goal can be achieved if the average group size, i.e., the ratio of row counts in the input and in the output, exceeds the degree of parallelism. A common name for this technique is "local-global aggregation." The initial, local aggregation may be opportunistic, i.e., it performs as much aggregation as can readily be achieved with the available memory, or it may be complete, i.e., it will spill overflow partitions to disk if required. If network transfer is faster than local disk I/O, overflow partitions may be send to their final destination rather than to a local disk.

Sort-based aggregation and duplicate elimination can benefit from the same ideas. The opportunistic variant performs merely run generation and early duplicate elimination prior to data transfer. The full variant performs complete local sort operations and all aggregation and duplicate elimination prior to the data transfer.

Order-preserving and merging re-partitioning operations may experience a deadlock, assuming flow control or bounded buffers in the data exchange operation. The essence of this deadlock is that producers need to send data in the order produced, whereas consumers need to consume data in the appropriate sort order. For example, if producer 1 has data only for consumer 1, and producer 2 has data only for consumer 2, then consumer 1 waits for data from producer 2 in order to advance its merge logic, yet producer 2 waits for consumer 2 to release flow control, etc. There

are several possible solutions for this problem. A typical deadlock avoidance strategy is to let producers send dummy records to all consumers, such that the consumers can advance their merge logic. A typical deadlock resolution strategy is to spill data to disk, effectively going beyond bounded buffers and relaxing flow control.

Parallel nested loops join, index nested loops join, and general nested iteration add further complexity to the data exchange operation. One issue is the number of threads: if the nested operation should run in 4 threads (due to four on-disk partitions, for example) yet the outer operation runs in three threads, will in inner operation actually run in 12 threads? If not, communication and synchronization need careful design, including avoidance or resolution of deadlocks. Moreover, invoking the inner operation for batches of outer correlation values rather than for individual outer rows may reduce communication costs as well as create optimization opportunities in the inner query execution plan.

Another technique to reduce communication costs in all matching operations is bit vector filtering. Sometimes thought to apply only to hash join, it also applies to merge join and nested iteration. If the two inputs for the matching operation are scanned or computed in two separate phases, the first such phase can populate a bit vector filter that the second phase can exploit to eliminate some items without further predicate evaluation and in particular without communication in parallel query execution. The bit vector filter might permit hash collisions or hash collisions might be prevented by some means, typically involving dynamic growth of the bit vector filter. In addition to the actual bit vector filter, it can prove useful to retain information about the minimal and maximal values.

In addition to expensive join and grouping operations, parallel execution and bit vector filtering can benefit set operations such as intersecting lists of row identifiers when exploiting multiple indexes on the same table for a query with a conjunctive predicate. Other index operations include union, difference, and join. Index join sometimes refers to joining indexes of two tables prior to fetching complete rows from either table and sometimes refers to joining indexes from a single table in order to obtain all columns required in a query and to avoid fetching rows one-by-one. Both kinds of index joins can benefit from parallel execution and bit vector filtering.

Some operations, however, cannot exploit parallel execution. The most notable example is a "top" operation, e.g., a query to find the three most productive sales people in the organization. While the "top" operation permits local-global computation with potentially tremendous reduction in serial computation and in communication, the final computation cannot benefit from multiple concurrent threads. An exception to this exception are "top" operations that apply to groups of records, e.g., a query looking for the three most productive sales people in each region.

Some algorithm implementations also exploit local parallelism, independent of parallel execution of the overall query execution plan. Asynchronous I/O (sequential read-ahead, single-page prefetch, and write-behind) are forms of local concurrent execution. Expensive operations, e.g., sort operations, may benefit from multiple threads, e.g., fitting records into a sort operation's workspace including initialization of a pointer array, sorting and rearranging the pointer array to represent the desired sort order, and forming on-disk runs from pointer arrays already sorted.

Parallel Execution beyond Queries

In addition to query execution, parallelism is also needed for large update operations and, perhaps most importantly, in utilities that scan or modify entire indexes, tables, or databases.

Parallel update operations are required in shared-nothing databases, simply because updates like filters are processed locally at each data site. Otherwise, if all update operations are relatively small, parallel update operations are not required for performance. However, update operations can be large, e.g., in bulk insertion and bulk deletion, also known as load, import, or roll-in and as roll-out.

A parallel update algorithm can follow the partitioning of stored data, it can assign disjoint key ranges within B-tree indexes to separate update threads, or it can assign threads to individual indexes. The latter approach is a special form of index-by-index updates (as opposed to row-by-row updates) as it permits the changes for each index to be sorted like the index for fast modification with read-ahead, etc.

Not only the actual database modifications but also other activities associated with updates can be parallel, including verification of integrity constraints, update propagation to materialized and indexed views, etc. Of course, verification of integrity constraints can be

parallel also during definition of new constraints. Definition of new materialized and indexed views creates an opportunity for parallelism both while computing the query defining the view and while loading the query result into the new data structures. Refreshing a view by re-computation is very similar.

In practice, however, the most important parallel operation is probably index creation. High performance index creation is particularly important if online index creation is not supported, i.e., an entire table is read-locked while the index utility scans, sorts, and inserts data. Fortunately, parallel versions of all three of these steps are readily available using standard mechanisms for parallel query execution.

Key Applications

Parallel algorithms for database query execution are used for two reasons. First, if there are more processors than active users, the only means by which these processors can do useful work on behalf of the users are parallel algorithms. Pure pipelining often leads to poor load balancing; thus, partitioning stored data and intermediate results, in particular using hash functions, often permits higher and more reliable speed-up. In the future, many-core processors may well increase the importance of parallel algorithms in database query execution.

Second, the hardware might require parallel algorithm because the database and its data are partitioned across multiple nodes each with its own memory, operating system, etc. If communication is more expensive than immediate data reduction (using selection, projection, and local aggregation), parallel algorithm are required. The future of improvements in processing bandwidth and in communication bandwidth will probably continue to be unbalanced; the current trend towards networked storage over direct-attached storage may reverse, in particular for database systems that support a single-system image over many direct-attached storage devices including management of that storage.

Future Directions

While processor speed kept increasing, the value of parallel query execution has been doubted at times. With hardware development now focusing on many-core processors rather than ever higher clock speeds, there should be no doubt that parallel query execution is required for relational data warehousing and business intelligence. It may be worth noting, however, that parallel utilities such as index creation are more urgently needed by customers with growing data volumes than parallel query execution. Many parallel utilities are implemented using mechanisms shared with parallel query execution, however, such that parallel query technology is needed in any case.

With the emergence of XML and semi-structured data in database type systems, there probably will be growing interest in parallel algorithms for relational algebra extended for unstructured data and graph manipulation. In addition, more data cleaning, data mining, and scientific applications are integrated with databases, and they, too, require parallel execution very similar to parallel query execution. Perhaps the future of parallel query execution is limited by advances in extensible query optimization, such that both together enable deep integration of those parallel algorithm into the query processing framework.

Cross-references

- ▶ Partitioning
- ▶ Query Optimization for Parallel Execution
- ▶ Relational Algebra
- ▶ Storage Resource Management
- ▶ Workload Management

Recommended Reading

1. Bratbergsengen K. Hashing methods and relational algebra operations. In Proc. 10th Int. Conf. on Very Large Data Bases, 1984, pp. 323–333.
2. DeWitt D.J., Gerber R.H., Graefe G., Heytens M.L., Kumar K.B., and Muralikrishna M. GAMMA – a high performance dataflow database machine. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 228–237.
3. Fushimi S., Kitsuregawa M., Tanaka, H. An overview of the system software of a parallel relational database machine GRACE. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 209–219.
4. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.

Parallel Query Optimization

Hans Zeller, Goetz Graefe
Hewlett-Packard Laboratories, Palo Alto, CA, USA

Synonyms

Optimization of parallel query plans

Definition

Parallel query optimization is the process of finding a plan for database queries that employs parallel hardware effectively. The details of this process depend on the types of parallelism supported by the underlying hardware, but the most common method is partitioning of the data across multiple processors.

Historical Background

Most parallel database systems today can trace part of their heritage back to the Gamma project at the University of Wisconsin, Madison in the 1980s – with the exception of the Teradata system, which predates Gamma by several years. Also influential were the GRACE database machine, developed at the University of Tokyo, and work at the Norwegian Institute of Technology, University of Trondheim. These projects did not publish a description of their parallel query optimization algorithms, however. Later projects, like XPRS (University of California, Berkeley) and IBM DB2 Parallel Edition describe that process in more detail.

A simple way of generating parallel query plans is to take the optimal plan produced by a conventional query optimizer and to parallelize it where possible. This may not result in the optimal parallel plan. For example, the chosen join order may require repartitioning of the data, while a slightly different join order might have avoided this costly step. Therefore, most of today's systems integrate parallelism into the optimization phase itself. Different parallel and serial plans are compared on the basis of their estimated cost, and the best plan is selected.

Most optimizer designers of parallel systems chose a very natural extension of non-parallel optimizers. This approach is described in the next section.

Foundations

Most of the parallel database systems use an operator-based model for parallel queries: Traditional relational operators such as join and groupby are employed in a parallel framework, allowing individual CPUs of a parallel system to work on a partition of the data. The task of the query optimizer is to find the right partitioning and degree of parallelism for each step.

With such an operator-based approach, optimizing parallel queries is similar to conventional query optimization, except that a new dimension is added – partitioning.

Extending a non-parallel optimizer to handle parallelism consists of four basic steps: First, the space of semantically correct parallel plans is defined. Second, the optimizer's search algorithm needs to be extended to enumerate a good part of the search space that was defined in the first step. Third, the cost model of the optimizer needs to be able to cost parallel plans such that the best one can be selected. Finally, all these extensions will likely increase the complexity of the optimization algorithm significantly. Heuristics are required to avoid an explosion of the cost of the optimization itself.

The following explores each of these four steps briefly.

Extending Query Plans to Execute in Parallel on Partitioned Data

The optimizer is responsible for generating semantically valid parallel plans. It therefore needs to have a good understanding of and model for parallel operator semantics. This is typically done by assigning a "partitioning" property to each relational operator and by using these properties to constrain the search space to semantically correct query plans. The partitioning property specifies a partitioning scheme (usually hash-based), the partitioning key, and the number of partitions.

In the following, the basic idea behind partitioning properties of a few key relational operators will be discussed.

When a file scan operator is executed in parallel, each parallel instance simply reads a partition of the data. On a shared-nothing system, the partitioning property describes how the data is physically partitioned. On a shared-everything architecture, another way is to let parallel scan operators compete for each block as it is read from disk, achieving a natural load balancing between the parallel operators, with a partitioning property that specifies only the number of partitions, not a partitioning key.

For joins, there are basically two different parallelization strategies. The first is to let each parallel instance join one partition of the first table R with a matching (co-located) partition of the second table S. Both R and S must have a matching partitioning property on the equi-join columns for this case. This is so that for a given row from a partition of R, all its matching rows from S are found in the corresponding partition of S.

The second parallel join algorithm is to join each partition of R with every partition of S or vice-versa. Such joins combine a partitioned table R with a

replicated table S or vice versa. This type of parallel join is applicable in nearly all cases, except for full outer joins.

Terms like "co-located nested loop join," "broadcast hash join" or "repartitioned merge join" are used to distinguish these different cases [13].

Semijoins and outer joins can be parallelized as well, with the exception that only the operand requiring special handling can be broadcast. Parallelizing the UNION operator puts no special constraints on the partitioning properties, except what is needed for duplicate elimination.

To increase its choices, the execution engine typically is capable to produce partitioning artificially. This is in most systems implemented by a separate operator, called Exchange [11], Table Queue (DB2), or River [2]. Generally, the exchange operator is the only one that sends data through messages, all other operators process local data and pass their results on in local memory or in a local disk file.

Extending the Search Algorithm to Include Parallel Plans

Almost all query optimizers operate under a principle borrowed from dynamic programming that combines optimal sub-solutions to a larger optimal solution. Sub-solutions are optimal with respect to relevant properties, as described first in [Selinger]. Probably the key aspect of query optimization for parallel queries is that this system of properties can be very elegantly extended to include partitioning properties. Initially only consisting of ordering, with a sort operator to create artificially sorted results, partitioning properties and the exchange operator are now included to create artificially partitioned results.

Extending the properties of query execution plans with partitioning solves the key part of parallel query optimization, namely the problem of enumerating the parallel plans that are possible for a query. The driver for this enumeration are the different ways the other operators can be executed in parallel.

The exchange operator acts as the "enforcer" [11] or "glue" [14] for partitioning.

This extension applies to classical Selinger-style optimizers (e.g., DB2) as well as to rule-based optimizers derived from the Exodus, Volcano and Cascades approaches (e.g., Microsoft SQL Server).

Costing Parallel Query Plans

Executing queries in parallel usually increases the amount of resources consumed, due to communication and synchronization overhead. Parallel execution is therefore not well-suited to minimize the resource cost of queries. Most parallel query optimizers try to minimize the time a query would take if executed in single-user mode on the system. When costing a parallel operator, they therefore compute the cost for a single partition only (assuming that the data is equally distributed over the partitions without skew) and add the cost for needed synchronization. With the goal of keeping the overall query response time low, it is usually not beneficial to reduce the degree of parallelism, therefore the optimizers consider only plans that use all the processing nodes of the system and compare them with a serial plan. The explosion of choices comes with different methods of partitioning and/or replicating data to allow for different types of join execution plans.

Heuristics to Reduce the Number of Parallel Plans Considered

One heuristic, the use of "interesting orders" [Selinger], can be used for partitioning as well. Optimizers with top-down exploration achieve this filtering effect naturally. Without some heuristics, the number of possible partitioning schemes could easily explode. For example, consider a co-located parallel merge join for the query "select * from R join S on R.a = S.a and R.b = S.b and R.c = S.c." Any ordering on a subset of columns (a,b,c), combined with a partitioning on some or all of the columns in this subset would constitute a valid set of properties for the join operator – assuming it is present for both tables R and S. Most implementations of optimizers will heuristically try only a small number of possibilities in this case.

Key Applications

Parallel computers allow database applications to scale, and optimization of parallel queries allows users to write applications without expending effort on parallelizing database queries. Parallel databases are the main application today where automatic parallelization of generic requests is performed successfully.

Future Directions

Challenges for future systems include adaptive systems that learn from poorly parallelized queries and that avoid data skew automatically, as well as automatic advisors and optimizers of physical designs for parallel databases.

Cross-references

- ▶ Parallel Query Execution Algorithms
- ▶ Query Optimization

Recommended Reading

1. Ballinger C and Fryer R. Born to be parallel. why parallel origins give teradata an enduring performance edge. IEEE Data Eng. Bull., 20(2):3–12, 1997.
2. Barclay T., Barnes R., Gray J., Sundaresan P. Loading databases using dataflow parallelism. ACM SIGMOD Rec. 23(4):72–83, 1994.
3. Baru C.K., Fecteau G., Goyal A., Hsiao H.-I., Jhingran A., Padmanabhan S., and Wilson W.G. An overview of DB2 parallel edition. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 460–462.
4. Boral H., Alexander W., Clay L., Copeland G.P., Danforth S., Franklin M.J., Hart B.E., Smith M.G., and Valduriez P. Prototyping Bubba, a highly parallel database system. IEEE Trans. Knowl. Data Eng., 2(1):4–24, 1990.
5. Bratbergsengen K. Algebra operations on a parallel computer – performance evaluation. In Proc. 5th Int. Workshop on Data Machines, 1987, pp. 415–428.
6. Chen A., Kao Y.-F., Pong M., Shak D., Sharma S., Vaishnav J., Zeller H. Query processing in nonstop SQL. IEEE Data Eng. Bull., 16(4):29–41, 1993.
7. DeWitt D.J., Gerber R.H., Graefe G., Heytens M.L., Kumar K.B., and Muralikrishna M. GAMMA – a high performance dataflow database machine. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 228–237.
8. DeWitt D.J. and Gray J. Parallel database systems: the future of high performance database systems. Commun. ACM, 35(6):85–98, 1992.
9. DeWitt D.J., Smith M., and Boral H. A single-user performance evaluation of the Teradata database machine. In Proc. of the 2nd Int. Workshop on High Performance Transaction Systems, 1987.
10. Fushimi S., Kitsuregawa M., and Tanaka H. An overview of the system software of a parallel relational database machine GRACE. In Proc. 12th Int. Conf. on Very Large Data Bases, 1986, pp. 209–219.
11. Graefe G. Encapsulation of parallelism in the volcano query processing system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990, pp. 102–111.
12. Graefe G. and Davison D.L. Encapsulation of parallelism and architecture-independence in extensible database query execution. IEEE Trans. Software Eng., 19(8):749–764, 1993.
13. Jhingran A., Malkemus T., and Padmanabhan S. Query optimization in DB2 parallel edition. IEEE Data Eng. Bull., 20(2):27–34, 1997.
14. Lee M.K, Freytag J.C., Lohman G.M. Implementing an interpreter for functional rules in a query optimizer. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988, pp. 218–229.
15. Mohan C. Pirahesh H., Tang W.G., and Wang Y. Parallelism in relational database management systems. IBM Sys. J., 33(2):349–371, 1994.
16. Neches P.M. The anatomy of a database computer system. In Digest of Papers - COMPCON, 1985, pp. 252–254.
17. Selinger, P., Astrahan, M., Chamberlin, D., Lorie, R. and Price, T. Access Path selection in a relational database management system. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1979, pp. 23–34.
18. Stonebraker M., Katz R.H., Patterson D.A., Ousterhout J.K. The design of XPRS. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988, pp. 318–330.
19. von Bueltzingsloewen G. Optimizing SQL queries for parallel execution. In Proc. on Workshop on Database Query Optimization, 1989.

Parallel Query Processing

Esther Pacitti[1,2]
[1]University of Nantes, Nantes, Cedex, France
[2]INRIA, LINA, Nantes, Cedex, France

Synonyms

Multiprocessor query processing

Definition

Parallel query processing designates the transformation of high-level queries into execution plans that can be efficiently executed in parallel, on a multiprocessor computer. This is achieved by exploiting the way data is placed in parallel and the various execution techniques offered by the parallel database system. As in query processing, the transformation from the query into the execution plan to be executed must be both correct and yield efficient execution. Correctness is obtained by using well-defined mappings in some algebra, e.g., relational algebra, which provides a good abstraction of the execution system. Efficient execution is crucial for high-performance, e.g., good query response time or high query throughput. It is obtained by exploiting efficient parallel execution techniques and query optimization which selects the most efficient parallel execution plan among all equivalent plans.

The main forms of parallelism which can be inferred by high-level queries are inter-query parallelism (several queries executed in parallel) and intra-query parallelism (each query executed in parallel), and within a query, inter- and intra-operator parallelism. These various forms of parallelism are obtained based on how data is placed in the parallel system, i.e., physically on disk or memory. Hence, the term data-based parallelism is communally used. These forms of parallelism can be combined and are exploited by parallel execution techniques.

Historical Background

Parallel query processing had been initiated in the context of database machines in the late 1970's. At that time, parallel processing techniques were already used successfully in scientific computing to improve the response time of numerical applications. However, these techniques are very complex as they typically involve parallelizing compilation which must infer parallelism from programs written in a procedural language, e.g., Fortran or C. Since these programs may be sequential, inferring parallelism to obtain good performance is hard.

With a high-level language like SQL, parallelism is relatively easier to infer. SQL enables programmers to specify what data to access (with predicates) without any detail on how. Thus, this provides the parallel database system much leverage to decide how to access data.

In the context of database machines, parallel query processing was very simple and concentrated on select-project-join queries using brute-force algorithms and special-purpose hardware. As parallel database systems evolved to exploit general-purpose multiprocessors and software-oriented solutions, parallel query processing has become much more complex. Also, the emergence of new applications such as OLAP and data mining translated into new features in SQL which have made parallel query processing more complex.

Compared to distributed query processing (which can also exploit parallelism by executing a query using multiple sites), the main difference is that the parallel system can have many more nodes and a very fast interconnection network. Thus, the execution techniques and execution plans are fairly different.

Foundations

Parallel query processing is the major solution to high-performance management of very large databases. The basic principle is to partition the database across multiple disks or memory nodes so that much inter- and intra-query parallelism can be gained. This can lead to significant improvements in both response time (the time the query takes to be executed and return results) and throughput (number of queries or transactions per time unit). However, decreasing the response time of a complex query through large-scale parallelism may well increase its total time (by additional communication) and hurt throughput as a side-effect. Therefore, it is crucial to optimize and parallelize queries in order to minimize the overhead of parallelism, e.g., by constraining the degree of parallelism for the query.

The performance of a parallel database system should ideally demonstrate two advantages: linear speedup and linear scaleup. *Linear speedup* refers to a linear increase in performance for a constant database size and linear increase in the number of nodes (i.e., processing and storage power). Thus, the addition of computing power should yield proportional increase in performance. *Linear scaleup* refers to a sustained performance for a linear increase in both database size and workload and number of nodes. Furthermore, extending the system should require minimal reorganization of the existing database. Thus, an increase of computing power proportional to the increase in database size and load should yield the same performance. These advantages are ideal since, in practice, increasing the configuration tends to increase the overhead of parallelism (e.g., more communications between nodes, more interference when accessing shared resources, etc.).

Partitioned data placement is the basis for the parallel execution of database queries. Given a parallel system with n nodes (each node may have one or more processors accessing a main memory and disks), each database table T is typically partitioned onto a number of nodes (less or equal to n) so that each node stores a subset of the tuples of T. This corresponds to horizontal fragmentation in distributed databases and similar techniques can be used. However, because there can be many nodes, more scalable techniques are also often used such as round robin or hashing on the placement attribute(s). Furthermore, to improve availability and performance, some partitions (typically those which are accessed more than others) can be replicated.

Given a partitioned database, parallel query execution can exploit two forms of parallelism: inter- and intra-query. *Inter-query parallelism* enables the parallel execution of multiple queries, each at a different node, in order to increase query throughput. This form of parallelism, reminiscent of that used in transaction processing systems, is quite simple since queries need not be parallelized. Thus, incoming queries can simply be dispatched (by a load balancer of the parallel database system) to the nodes which store the data corresponding to the queries. However, this can only work if all the data accessed by a query are stored at the node, or accessible at the node. For instance, in the case of a shared-nothing parallel database

system, if a query Q involves data that is partitioned at two different nodes, then either node cannot entirely execute the query. But in the case of a shared disk parallel database system, since all the disks can be accessed from each node, the same query Q could be executed by either node.

A query can be decomposed in a tree of relational operators, where each operator takes data as input (either base data or temporary data) and produces temporary data, with the root operator producing the final result. This decomposition allows for *intra-query parallelism*) which has two forms: inter-operator and intra-operator parallelism. *Inter-operator* parallelism is obtained by executing in parallel several operators of the query on several nodes while with intra-operator parallelism, the same operator is executed by many nodes, each one working on a different partition of the data. For inter-query parallelism, much attention has been devoted to pipelined (or consumer-producer) parallelism which enables consumer operators to start execution as soon they get input data from producer operators. Pipelined executions do not require temporary relations to be materialized, i.e., a tree node corresponding to an operator executed in pipeline is not stored. These two forms of parallelism complement each other well since inter-operator parallelism gets beneficial as the query gets complex (with many operators) while intra-operator parallelism gets beneficial for heavy operators (which access large parts of the data).

To exploit inter- and intra-query parallelism, using database partitioning, parallel algorithms for the execution of relational operators are necessary. Such algorithms must yield a good trade-off between parallelism and communication cost since increasing parallelism involves more communication among nodes. Parallel algorithms for unary operators (e.g., select) are relatively simple as they typically exploit data placement or secondary indices to restrict access to base data. Parallel algorithms for binary operators such as join are much more involved since join can incur much communication. The main algorithms exploit the placement of one of the two relation to reduce communication or artificially create a good data placement using hashing. When there is sufficient main memory to hold one of the two relations (which is a very practical case), the execution of hash-based join can be pipelined across multiple operator nodes, thus increasing performance. Since the intermediate results of the operators can be skewed, parallel algorithms must also deal with load unbalancing, e.g., by redistributing a heavy work at one node to multiple nodes using hashing.

Parallel query optimization is the process of selecting the best parallel execution plan for a query. Compared to distributed query optimization, it focuses on taking advantage of both intra-operator parallelism and inter-operator parallelism. As any query optimizer, a parallel query optimizer can be seen as three components: a search space, a cost model, and a search strategy. Parallel execution plans are abstracted by means of operator trees, which define the order in which the operators are executed. Operator trees are enriched with annotations, which indicate additional execution aspects, such as the algorithm of each operator. An important aspect to be reflected by annotations is that operators are executed in pipeline. The cost model provides the cost functions for parallel operator algorithms.

Key Applications

Parallel query processing has been developed in the context of parallel database systems, with a focus on OLAP applications, where good response time is crucial. Most of the work has been done in the context of the relational model. In order to support new OLAP applications which may access all kinds of data, including unstructured and semi-structured (XML) data, major extensions to parallel query processing are necessary. In particular, techniques from parallel database and information retrieval need to be combined.

Cross-references

- ▶ Distributed Databases
- ▶ Parallel Data Placement
- ▶ Parallel Query Execution Algorithms
- ▶ Query Optimization

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2):73–170, 1993.
2. Kossmann D. The state of the art in distributed query processing. ACM Comput. Surv., 32(4):422–469, 2000.
3. Lanzelotte R., Valduriez P., Zait M., and Ziane M. Industrial-strenght parallel query optimization: issues and lessons. Inf. Sys., (19)4:311–330, 1994.
4. Özsu M.T. and Valduriez P. Principles of distributed database systems. Prentice-Hall, 2nd edn, 1999.
5. Valduriez P. and Pacitti E. Parallel database systems. In Handbook of Database Technology, J. Hammer, M. Scheider (eds.). CRC, USA, 2007.

Parallel SCSI

► Storage Protocols

Parameterized Complexity of Queries

Christoph Koch
Cornell University, Ithaca, NY, USA

Definition

Parameterized complexity theory is the study of the interaction between the fixing of parameters of input problems and their computational complexity. A central parameterized complexity concept is that of a fixed-parameter tractable (FPT) problem, which captures a strong notion of well-behavedness of a problem under the assumption that parameter values do not grow with input sizes. There is also a solid theory of fixed-parameter intractability, which gives strong evidence that for certain parameterizations of problems, no FPT algorithms can be found.

Historical Background

Fixed-parameter complexity theory is strongly associated with R. Downey and M. Fellows, who did much seminal work in the area (cf. [3,5]). The first fixed-parameter complexity result in the context of database query evaluation was the linear-time query processing algorithm for acyclic conjunctive queries by Yannakakis in 1981 [12], which preceded the development of parameterized complexity theory (cf. also [13]).

Foundations

Fixed-Parameter Tractability

A *parameterized (decision) problem* is a set of pairs (x,k), where x is called the input and k the parameter (an integer). As a convention, $n = ||x||$ will be used to denote the size of the input. A parameterized problem is called strongly uniformly *fixed-parameter tractable* (subsequently, just fixed-parameter tractable, or FPT), if there is a computable integer function f, a constant c, and an algorithm that, given a parameterized problem instance (x, k), decides x in time $O(f(k) \cdot n^c)$.

It is important to note the difference between an algorithm that runs in time $O(n^k)$ and an FPT algorithm. If $k = 10$ and $c = 1$, an algorithm that runs in time $O(f(10) \cdot n)$ is linear (with possibly a large constant), while an $O(n^{10})$ algorithm will not even scale to very moderately sized problem instances.

Consider the *data complexity* of queries [11], i.e., the complexity of evaluating a query on a database if only the database is considered part of the input, while the query is fixed and part of the problem specification. It is well known that quite naive query evaluation techniques can evaluate an arbitrary relational algebra query in time $O(n^k)$, where k is the size of the query (e.g., the number of algebra operators involved). By fixing k, the query evaluation time becomes polynomial, but the $O(n^k)$ time query evaluation technique does not yield a fixed-parameter tractability result. Indeed, such an FPT result is considered unlikely to exist.

For another example, consider the Set Cover problem, a well known combinatorial problem that appears in database problems such as subspace clustering or finding pipelined query plans in stream processors.

A set C is called a *cover* of a set of sets $\mathbf{S}$ if for each $S \in \mathbf{S}$, $S \cap C \neq \emptyset$. The Set Cover problem is defined as follows.

Set Cover
Input: An integer k, a finite set V, and a set $\mathbf{S} \subseteq 2^V$ of subsets of V.
Question: Is there a set C with $|C| \leq k$ such that C is a cover for $\mathbf{S}$?

The Set Cover problem is NP-complete. A naive brute-force algorithm would check for each set $C \subseteq V$ with $|C| \leq k$ whether C is a cover. This algorithm runs in time $O(|V|^k \cdot n)$, which is polynomial in n if k is fixed. However, this is not an FPT algorithm with parameter k because $|V|$ may get arbitrarily large with the input.

Consider the following refined algorithm for Set Cover (cf. [5,8]).

$\mathbf{C}_0 := \{\emptyset\}$;
for $i = 1$ **to** k **do** $\mathbf{C}_i = \{C \cup \{a\} | C \in \mathbf{C}_{i-1}, a \in S(C)\}$;
if at least one element of $\mathbf{C}_k$ is a cover for $\mathbf{S}$ **then output** true;
else output false

Here, $S(C)$ is a function that returns, in a deterministic way, an element S of $\mathbf{S}$ such that, if this condition can be satisfied by any element of $\mathbf{S}$, $S \cap C \neq \emptyset$. One way of defining $S(C)$ is as

$$S(C) = \begin{cases} \min \mathbf{S}(C) \ldots \mathbf{S}(C) \neq \emptyset \\ \min \mathbf{S} \ldots \text{otherwise.} \end{cases}$$

where $\mathbf{S}(C) = \{S \in \mathbf{S} \mid S \cap C \neq \emptyset\}$ and min returns the smallest element of a subset of $\mathbf{S}$ with respect to some arbitrary fixed order among the elements of $\mathbf{S}$ (e.g., the order in which the elements of $\mathbf{S}$ are stored in memory).

Consider for example $\mathbf{S} = \{\{a, b\},\{a, c\},\{b, d\},\{c, e\}, \{c, f\}\}$ with the elements of $\mathbf{S}$ ordered as just enumerated. Then, $S(\emptyset) = \{a, b\}$, $\mathbf{C}_1 = \{\{a\},\{b\}\}$, $S(\{a\}) = \{b, d\}$, $S(\{b\}) = \{a, c\}$, and $\mathbf{C}_2 = \{\{a, b\},\{a, d\},\{b, c\}\}$. Since $\{b, c\}$ is a cover, the algorithm returns true.

It is easy to check that this algorithm runs in time $O(s^k \cdot n)$ where s is the maximum cardinality among the elements of $\mathbf{S}$, i.e., $s = \max\{|S| : S \in \mathbf{S}\}$. Thus, Set Cover with parameter $k + s$ (which is not the standard parameterization for Set Cover, which would be k) is FPT. If s is small, this may be a useful algorithm.

Fixed-Parameter Intractability

For a number of parameterized versions of NP-hard problems it can be shown that they are either provably not FPT or not FPT unless P=NP. An example for the former scenario would be EXPTIME Turing machine acceptance (which is EXPTIME-complete) with a dummy parameter that is unrelated to the input. An example of the latter scenario is the satisfiability of CNF formulae whose clauses have no more than k literals, with parameter k. For $k = 3$, this is the 3SAT problem, which is NP-complete. But there are also parameterized problems for which apparently no FPT algorithm exists and for which more subtle hardness arguments based on special complexity classes for parameterized problems have to be developed.

A *fixed-parameter many-one reduction* is a reduction that maps each instance (x, k) of a parameterized problem to an instance (x', k') of another parameterized problem using an FPT algorithm (i.e., in time $O(f(k) \cdot x^c)$), with the additional condition that the size of k' must only depend on k but not on x, such that (x, k) is a yes-instance of the first problem if and only if (x', k') is a yes-instance of the second problem. Closure under such reductions yields robust complexity classes. The class of parameterized problems that are fixed-parameter many-one reducible to problem Π shall be denoted by $[\Pi]^{\text{fpt}}$. A problem in $[\Pi]^{\text{fpt}}$ is called *complete* for $[\Pi]^{\text{fpt}}$ if Π is fixed-parameter many-one reducible to it.

Let $\Gamma_{0,d} = \Delta_{0,d}$ be the class of all propositional formulae constructible from propositional variables using negation and the *binary* operations conjunction $\wedge$ and disjunction $\vee$ such that the maximum expression depth, not taking into account negations, is d. In other words, for any such formula, the maximum number of disjunctions or conjunctions occurring on paths from the root of the expression tree to a leaf is d. Now, $\Gamma_{t,d}$ consists of the formulae of the form $\wedge\ \Phi$, where Φ is a finite subset of $\Delta_{t-1,d}$, and $\Delta_{t,d}$ consists of the formulae of the form $\vee\ \Psi$, where Ψ is a finite subset of $\Gamma_{t-1,d}$.

A usual way of defining the W hierarchy is in terms of a weighted version of the propositional satisfiability problem. The weight of a truth assignment for the variables of a propositional formula is the number of variables set to true in that truth assignment. The weighted satisfiability problem $\text{WSAT}(\Gamma_{t,d})$ for the class of formulae $\Gamma_{t,d}$ is as follows: Given a formula $\phi \in \Gamma_{t,d}$ and parameter k, does ϕ have a satisfying assignment of weight k?

The W-hierarchy, for each $t \geq 1$, is defined as

$$W[t] := \bigcup_{d \geq 0} \left[\text{WSAT}(\Gamma_{t,d})\right]^{\text{fpt}}$$

with $W[t] \subseteq W[t+1]$ and is believed to be strict ($W[t] \subsetneq W[t+1]$). For a few examples, natural parameterized versions of Clique and Independent Set are $W[1]$-complete, while parameterized versions of Hitting Set, Dominating Set, and Kernel are $W[2]$-complete (see [5]). These results depend on the choice of parameter, and the standard parameterization is the size of the structure (set) whose existence is to be guessed and verified. The question whether FPT $\neq$ W[1] is the parameterized complexity analogue of the question whether P $\neq$ NP. It remains unproven, but is strongly suspected.

Returning to the evaluation complexity of relational algebra queries with the size of queries as the parameter,

Parameterized Query Evaluation
Input: A Boolean relational calculus query Q and a relational database.
Parameter: The size k of a reasonable representation of Q in bits.
Question: Does Q return true on the input database?

which is known to be $W[1]$-complete [9] already for conjunctive queries (i.e., select-project-join queries) and AW[*]-complete for full relational calculus [4]. AW[*] is a complexity class that subsumes the $W[t]$ classes, for all t, but may contain additional problems.

Further Positive Results on Query Evaluation Complexity

While relational query languages such as relational algebra, calculus, and datalog (a generalization of conjunctive queries) are W[1]-hard and thus unlikely to be fixed-parameter tractable, there are important fragments of these languages that are FPT. Moreover, there are other logics and query languages, specifically on tree- and graph-structured data models, which are FPT with the size of the query as the parameter.

The first FPT result in the context of database query processing was on the evaluation of *acyclic* conjunctive queries by Yannakakis. Consider a Boolean conjunctive query in datalog notation, $q \leftarrow R_1(\vec{y}_1),\ldots,R_k(\vec{y}_k)$. The acyclicity of queries refers to a notion of acyclicity of associated hypergraphs, whose nodes are query variables and whose hyperedges are the sets of variables $\vec{y}_i$ occurring together in an atomic formula of the query. Acyclicity can be defined in a number of ways, such as by a low-complexity algorithm or using guarded logic. An exact definition is beyond the scope of this article, but in the case that all input relations are binary and no atom is of the form $R(y, y)$, the hypergraph is an undirected graph and hypergraph acyclicity coincides with the standard graph-theoretic notion of acyclicity. By exploiting the tree structure of this graph, which describes the necessary joins, and projecting away columns that will not be involved in further joins as early as possible, it is always possible to find a query plan that can be evaluated in time $O(f(k) \cdot n)$ where k is the size of the query – i.e., the problem is fixed-parameter linear.

A Boolean acyclic conjunctive query can be written using just selections and semijoin operations, with a $\pi_\emptyset$ operation on top. If Q_1 and Q_2 are select-semijoin-queries, then $Q_1(x) \ltimes Q_2(x) = \pi_{\mathsf{sch}(Q_1)}(Q_1(x) \bowtie Q_2(x))$ is a subset of $Q_1(x)$ and can be computed in linear time in $|Q_1(x)| + |Q_2(x)|$, where x is the input database. By induction it follows that overall query evaluation is FPT.

Consider the acyclic conjunctive query $\pi_\emptyset\ (R \bowtie S \bowtie T \bowtie U \bowtie V)$ for database schema $R(A,B), S(B,C), T(C,D), U(C,E), V(A,F)$. The hypergraph (here, graph) is acyclic:

The query plan $\pi_\emptyset\ ((R \ltimes V) \ltimes ((S \ltimes \mathbf{T}) \ltimes U))$ admits efficient evaluation.

For nonboolean conjunctive queries, the running time is polynomial of degree c where c is the arity of the query result (which must be considered a true constant rather that a parameter if this is to be considered an FPT result), and clearly, this is optimal because the output size of a query that simply computes the product of the input relations is $\Omega(n^c)$.

The notion of hypergraph acyclicity has been generalized to queries of bounded hypertree-width, which is a measure of how tree-like the query hypergraph is: hypertree-width 1 coincides with hypergraph acyclicity. It was shown in [7] that conjunctive query evaluation with the hypertree-width of the query as the parameter is FPT and that this generalizes in a natural way to relational calculus queries. The fixed-parameter tractability of queries of bounded hypertree-width > 1 (in fact, 2) has been used to explain the polynomial-time complexity of XPath queries [1].

A classical result by Courcelle [2] shows that very powerful queries – in monadic second-order logic, a language that strictly subsumes relational calculus – over databases of bounded tree-width (the exact definition is technical, but in the case that all relations are binary, the condition is bounded tree-width of the (undirected) graph obtained by unioning the relations together) are fixed-parameter linear. Note that in this result the structure of the data is restricted but the structure of queries is not. The special case where the database is a tree with node labels from a finite alphabet is due to Doner, Thatcher and Wright [10]. The algorithm is based on compiling the query into a tree automaton which, once obtained, can be evaluated on the data tree in linear time. Thus, the running time is $O(f(k) + n)$, where k is the size of the query. This is a famous case where f is much worse than singly exponential: f is nonelementary – a tower of twoes $2^{2^{2^{\cdot^{\cdot^{\cdot}}}}}$ whose height grows with k. This is apparently necessarily so: Frick and Grohe [6] show that unless $P = NP$, any FPT algorithm for the problem must have a nonelementary f.

Key Applications

Fixed-parameter complexity results can assist designers of data management algorithms in two ways. There is now a large set of positive results for a variety of parameterized versions of NP-hard problems, which may surface in contexts such as query optimization and data mining. For a particular data management

problem, it may be known that a particular parameter is always bounded in the problem instances that arise. If the problem is FPT for that parameter, there is an efficient algorithm for solving the problem. Furthermore, FPT results exist for fragments of the relational query languages such as relational algebra as well as for query languages for tree- and graph-structured data.

Conversely, if it is known that a parameterized problem is W[1]-hard, then it is quite hopeless to try to develop an efficient algorithm for solving the parameterized problem; in that case one may look for different acceptable parameterizations or for efficient approximation techniques.

Cross-references

- ▶ Complexity
- ▶ Complexity Results
- ▶ Tree Automata

Recommended Reading

1. Benedikt M. and Koch C. XPath Leashed. ACM Comput. Surv., 4(1), 2008.
2. Courcelle B. Graph rewriting: an algebraic and logic approach. In Handbook of Theoretical Computer Science, J. van Leeuwen (ed.). vol. 2, chap. 5, Elsevier B.V., Amsterdam, The Netherlands, 1990, pp. 193–242.
3. Downey R.G. and Fellows M.R. Parameterized Complexity. Springer, Berlin, 1999.
4. Downey R.G., Fellows M.R., and Taylor U. The parameterized complexity of relational database queries and an improved characterization of W[1]. In Proc. DMTCS '96. Combinatorics, Complexity, and Logic, 1996, pp. 194–213.
5. Flum J. and Grohe M. Parameterized Complexity Theory. Springer, Berlin, 2006.
6. Frick M. and Grohe M. The complexity of first-order and monadic second-order logic revisited. In Proc. 17th Annual IEEE Symp. on Logic in Computer Science, 2002, pp. 215–224.
7. Gottlob G., Leone N., and Scarcello F. Hypertree decompositions and tractable queries. J. Comput. Syst. Sci., 64(3):579–627, 2002.
8. Grohe M. Parameterized complexity for the database theorist. ACM SIGMOD Rec., 31(4), 2002.
9. Papadimitriou C.H. and Yannakakis M. On the complexity of database queries. In Proc. 16th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1997.
10. Thatcher J. and Wright J. Generalized finite automata theory with an application to a decision problem of second-order logic. Math. Syst. Theory, 2(1):57–81, 1968.
11. Vardi M.Y. The complexity of relational query languages. In Proc. 14th Annual ACM Symp. on Theory of Computing, 1982, pp. 137–146.
12. Yannakakis M. Algorithms for acyclic database schemes. In Proc. of the Seventh Int. Conf. on Very Large Data Bases, 1981, pp. 82–94.
13. Yannakakis M. Perspectives on database theory. In Proc. of the 36th IEEE Symp. on Foundations of Computer Science, 1995, pp. 224–246.

Parametric Data Reduction Techniques

Rui Zhang
University of Melbourne, Melbourne, VIC, Australia

Definition

A parametric data reduction technique is a data reduction technique that assumes a certain model for the data. The model contains some parameters and the technique fits the data into the model to determine the parameters. Then data reduction can be performed.

Key Points

Parametric data reduction (PDR) techniques is opposite to nonparametric data reduction (NDR) techniques. A model with parameters is used in a PDR technique and therefore some computation is required to determine these parameters, which may be costly. However, if a PDR technique is well-chosen, it may result in much more data reduction than NDR techniques. A representative example is linear regression [3]. Linear regression assumes that the data fall on a straight line, expressed by the following formula

$$Y = a + bX \tag{1}$$

Given a set of points (Assuming two dimensions.) $\{\langle x_1, y_1\rangle, \langle x_2,y_2\rangle,...\}$, parameters a and b in Equation (1) are determined from the points using the least squares criteria. The result is

$$b = \frac{\sum(X - \overline{X})(Y - \overline{Y})}{\sum(X - \overline{X})^2}$$

$$a = \overline{Y} - b\overline{X}$$

where $\overline{X}$ and $\overline{Y}$ are the average values of $x_1,x_2,...$and $y_1,y_2,...$, respectively. If the data are actually distributed on almost a line, linear regression is a very efficient data reduction technique. Besides the values of a and b,

only one dimension of every point is needed to represent the data set. The data volume is reduced by half. However, if the data are not distributed on a line, linear regression will result in large errors.

Other popular PDR techniques include Singular Value Decomposition (SVD) [2] and Discrete Wavelet Transform (DWT). SVD assumes that a matrix is decomposed to the form of $\mathbf{A} = \mathbf{USV}^t$ and $\mathbf{U}, \mathbf{S}, \mathbf{V}$ need to be determined; DWT assumes that a signal is projected to a set of orthogonal basis vectors and the wavelet coefficients need to be determined. A summary of data reduction techniques including PDR techniques can be found in [1].

Cross-references

- ▶ Data Deduction
- ▶ Discrete Wavelet Transform and Wavelet Synopses
- ▶ Linear Regression
- ▶ Nonparametric Data Reduction Techniques
- ▶ Singular Value Decomposition

Recommended Reading

1. Barbará D., DuMouchel W., Faloutsos C., Haas P.J., Hellerstein J.M., Ioannidis Y.E., Jagadish H.V., Johnson T., Ng R.T., Poosala V., Ross K.A., and Sevcik K.C. The New Jersey data reduction report. IEEE Data Eng. Bull., 20(4):3–45, 1997.
2. Jolliffe I.T. Principal Component Analysis. Springer, Berlin, 1986.
3. Wonnacott R.J. and Wonnacott T.H. Introductory Statistics. Wiley, New York, 1985.

Partial Replication

Bettina Kemme
McGill University, Montreal, QC, Canada

Definition

A replicated database consists of a set of nodes $\mathcal{N}$ (database servers) and each logical data item x has a physical copy on a subset $\mathcal{N}_x$ of the nodes in $\mathcal{N}$. The replication degree $r_x = |\mathcal{N}_x|$ of a data item is the number of copies it has. Using *full replication*, each logical data item has a copy on each of the nodes, i.e., for each data item x of the database, $\mathcal{N}_x = \mathcal{N}$. Whenever there is at least one data item that does not have copies at all nodes, one refers to a partial replication architecture.

Key Points

Partial Replication can be used for different purposes:

Cluster Replication

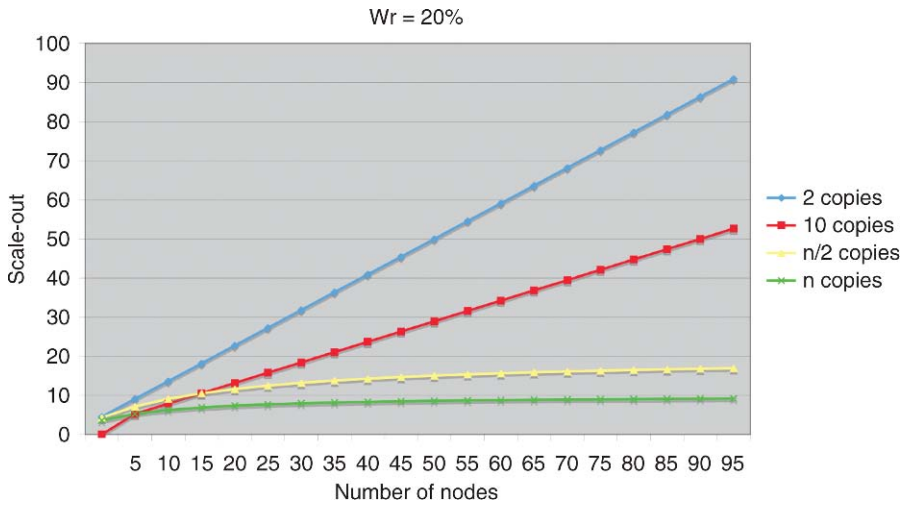
In order to achieve scalability, data can be replicated across nodes residing in a single cluster. Read access to data items can then be distributed across the existing copies. Write operations, however, have to be performed on all copies. The potential for scalability can be best understood by a simple analytical model. It assumes each transaction accesses one data item and has execution cost normalized to one unit. A fraction of wr are update transactions. Communication costs and concurrency control issues are ignored. A read-only transaction is executed at any node with a copy of the data item while an update transaction is executed at all nodes. The system consists of n nodes each having the capacity to process C execution units per time unit. There are m data items each having the same replication factor r. The copies are equally distributed across the nodes ($\frac{m*r}{n}$ copies per node). All data items are accessed with the same frequency and requests are equally distributed across all nodes.

If the replicated system can execute l transactions per unit then $(1 - wr) * l$ are read-only transactions executed at one node and $wr * l$ are update transactions executed at r nodes. Thus, the total capacity of the system is used as follows: $n * C = (1 - wr) * l + r * wr * l$. The scale-out is the number of transactions l the replicated system can handle per time divided by the number of transactions a non-replicated system can handle (which is C). Therefore, the scale-out of an n-node system is $\frac{n}{1+(r-1)*wr}$.

Figure 1 shows the scale-out for systems up to 100 nodes with an update load of 10%, and each data item has 2, 10, $n/2$ or n copies. While a constant replication factor provides linear scalability (and the smaller the better), a replication factor that increases with the number of nodes leads to a scalability cealing. Once the saturation point is reached, adding more nodes will not increase the throughput because applying update transactions consumes most of the available resources. More detailed performance models are given in [2,1].

WAN Replication

Replicating data items at different geographical locations is mainly used to provide fast local access to clients of the different regions. Having a data item replicated at a specific location decreases



Partial Replication. Figure 1. Scale-out of partial replication.

communication costs for read operations but increases communication and processing costs for update transactions, and has additional storage costs. Thus, placement algorithms have to decide where to put replicas in order to find a trade-off between the different factors [3].

Challenges

When a transaction executes at a specific node but the node does not have a copy of a requested data item, remote access is necessary. This poses several challenges. First, for update transactions concurrency control becomes more complicated. Second, a location mechanism must be implemented that finds nodes with appropriate data copies. Third, for complex SQL queries, query execution becomes distributed potentially requiring data shipping and advanced operators. Therefore, in practice, data that is often accessed together (within a transaction or query) is co-located. This, however, reduced the flexibility in terms of optimizing load-balancing and update processing.

Cross-references

- ▶ Replica Control
- ▶ Replication for Scalability
- ▶ WAN Data Replication

Recommended Reading

1. Nicola M. and Jarke M. Performance modeling of distributed and replicated databases. IEEE Trans. Data Knowl. Eng., 12(4):645–672, 2000.
2. Serrano D., Patiño-Martínez M., Jiménez-Peris R., and Kemme B. Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. In Proc. IEEE Pacific Rim Dependable Computing Conference, 2007.
3. Wolfson O., Jajodia S., and Huang Y. An adaptive data replication algorithm. ACM Trans. Database Syst. 22(2): 255–314, 1997.

Partitioned Query Execution

- ▶ Parallel Query Execution Algorithms

Partitioning

- ▶ Parallel Query Execution Algorithms

Passage Retrieval

- ▶ Structured Document Retrieval

Path Functional Dependencies

- ▶ Functional Dependencies for Semi-structured Data

Path Index

► Structure Indexing

Path Query

Yuqing Wu
Indiana University, Bloomington, IN, USA

Synonyms

Document path query

Definition

Given a semi-structured data set D, a *path query* identifies nodes of interest by specifying the *path* lead to the nodes and the predicates associated with nodes along the *path*. The *path* is identified by specifying the labels of the nodes to be navigated and structural relationship (parent-child or ancestor-descendant) among the nodes. A predicate can be a path query itself, relative to the node that it is associated with.

Historical Background

Using path information in query processing has been studied in the object-oriented database systems, in which most queries require the traversing from one object to another following object identifiers, in the mid 1990's. The notion of path query, in which the path and predicates along the path are specified as the core of the query, became popular with the growth of the information on the web and the introduction of semi-structured data, especially XML.

Most of the popular query languages for querying XML data, such as XPath [6] and XQuery [4], employ path query as the approach to identify nodes of interest. However, some techniques, such as schema-free XQuery, relax the requirements of specifying the exact path, but rely more heavily on the database management systems to detect the least common ancestors of the keywords specified in the query.

Algebraic research has been fruitful in studying, comparing and characterizing fragments of path queries [2,9], as well as methods and techniques for optimizing and evaluating path queries.

Foundations

Semi-structured data, for example, XML, consists of data entries and containment relationships among the data entries. The data, usually referred to as *a document*, is frequently represented by an ordered node-labeled tree or graph, depending on whether only the containment relationships are treated as first-class relationship, or the id-reference relationships among the data entries are also treated as first-class relationships. The tree representation is more popular:

A *document* D is a 3-tuple (V, Ed, λ), with V the finite set of nodes, $Ed \subseteq V \times V$ a set of parent-child edges, and λ: $V \rightarrow \mathcal{L}$ a node-labeling function into a countably infinite set of labels $\mathcal{L}$.

In addition, even though not always substantial, the ordering among siblings is usually an important feature for nodes in a *document*. The pre-order among the data entries is called the *document order*. Among others, the pre-order is a dominant approach used to identify data entries in a document, while the document is stored in a database system, relational or native.

The aim of the path query is to express the precise requirement of retrieving a set of data entries that satisfy certain value and structural requirements.

The algebraic form of a path query consists of the primitives of $\emptyset$, ε, $\widehat{l}(l \in \mathcal{L})$ for token matching, $\uparrow$ and $\downarrow$ for upward and downward navigation, and operations $\diamond$ for composition of two algebraic expression $E_1 \diamond E_2$, Π_1 and Π_2 for the first and second projection of an algebraic expression, and set operations $\cap$, $\cup$, and $-$. Given a document $D = (N, Ed, \lambda)$ and a path query E, the path semantics of $E(D)$ is a binary relation:

$$
\begin{aligned}
\emptyset(D) &= \emptyset \\
\epsilon(D) &= (n, n) \mid n \in N\} \\
\widehat{l}(D) &= \{(n, n) \mid n \in N \text{ and } \lambda(n) = l\} \\
\downarrow(D) &= Ed \\
\uparrow(D) &= Ed^{-1} \\
\Pi_1(E)(D) &= \pi_1(E(D)) \\
\Pi_2(E)(D) &= \pi_2(E(D)) \\
E_1 \diamond E_2(D) &= \pi_{1,4}\sigma_{2=3}(E_1(D) \times E_2(D)) \\
E_1 \cap E_2(D) &= E_1(D) \cap E_2(D) \\
E_1 \cup E_2(D) &= E_1(D) \cup E_2(D) \\
E_1 - E_2(D) &= E_1(D) - E_2(D)
\end{aligned}
$$

These primitives and operations identify the path of navigation in a document to reach the resultant data entries.

The $\uparrow^*$ and $\downarrow^*$ are frequently used to identify the ancestor-descendant relationship among data entries along a path:

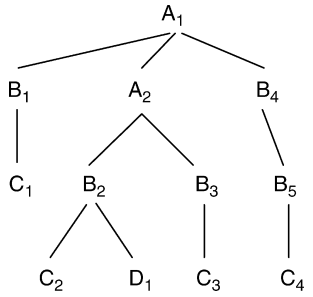
$$\downarrow^*(D) = \bigcup_{i=0..height(D)} \underbrace{\downarrow \ldots \downarrow}_{i}(D)$$

$$\uparrow^*(D) = \bigcup_{i=0..height(D)} \underbrace{\uparrow \ldots \uparrow}_{i}(D)$$

The result of a path query against document D under the *path semantics* is a set of node pairs whose neighborhood data entries and structures satisfy the query expression.

A localized semantics, also called the *node semantics* of a path query Q is to apply the path expression to a document D and a specific node n_0 in the document, such that $Q(D, n_0) = \{n|(n_0, n) \in Q(D)\}$. Usually, the results are presented in a list that honors the *document order*.

For example, assuming document D is represented as a tree structure as shown in Fig. 1, some sample path queries are evaluated as follows:



Path Query. Figire 1. A sample semi-structured document presented in a tree structure (subscripts are used to distinguish data entries with the same label).

parent-child or ancestor-descendant relationship) represent the structural requirements among the data entries.

The evaluation of a path query is also called the process of *pattern matching*, which is to find the *witness trees* that consist of data entries that satisfy both the node and structural constraint expressed by the pattern tree.

Path Query	Sample Result
$Q_1 = \downarrow \diamond B$	$Q_1(D) = \{(A_1, B_1), (A_1, B_4), (A_2, B_2), (A_2, B_3), (B_4, B_5)\}$ $Q_1(D, A_1) = \{B_1, B_4\}$
$Q_2 = \Pi_1(\downarrow \diamond \downarrow \diamond C)$	$Q_2(D) = \{(A_1, A_1), (A_2, A_2), (B_4, B_4)\}$ $Q_2(D, A_2) = \{A_2\}$ $Q_2(D, B_4) = \{B_4\}$
$Q_3 = \epsilon \diamond \Pi_1(\downarrow \diamond \downarrow \diamond C) \diamond \downarrow \diamond B \diamond \downarrow$	$Q_3(D) = \{(A_1, C_1), (A_1, B_5), (A_2, C_2), (A_2, D_1), (A_2, C_3), (B_4, C_4)\}$ $Q_3(D, A_1) = \{C_1, B_5\}$ $Q_3(D, B_4) = \{C_4\}$ $Q_3(D, B_2) = \emptyset$
$Q_4 = \uparrow \diamond \Pi_1(\downarrow^* \diamond D) \diamond \Pi_1(\downarrow \diamond \Pi_1(\downarrow \diamond B))$	$Q_4(D) = \{(B_1, A_1), (A_2, A_1), (B_4, A_1)\}$ $Q_4(D, B_1) = \{A_1\}$
$Q_5 = \epsilon \diamond \Pi_1((\downarrow \diamond B) \cup (\uparrow \diamond A)) \diamond \downarrow \diamond \downarrow$	$Q_5(D) = \{(A_2, C_2), (A_2, D_1), (A_2, C_3)\}$ $Q_5(D, A_2) = \{C_2, D_1, C_3\}$

Path Query and Pattern Tree Matching

Path queries are frequently represented as tree structure, called *pattern trees*, in which nodes (optionally labeled) represents the query requirement on the data entries along the path, and edges (labeled with

Path Query Languages

Path query is the core of various query languages for semi-structured data.

XPath [6] is a W3C standard for addressing part of an XML document and for matching and testing

whether a node satisfies a pattern. The primary syntactic construct in XPath is the expression, which is evaluated to yield a node set, a boolean value, a number, or a string. The core of the XPath query language is the path query, which is called *location paths* in XPath. Location path consists of relative location paths and absolute location path. A relative location path consists of a sequence of one or more location steps separated by "/". A step is composed from left to right and each step selects a set of nodes relative to a context node, which will in turn serve as the context node for the following step. An absolute location path consists of a leading "/", followed optionally by relative location paths.

XPath, in turn, is the core of other XML query languages and transformation languages, such as XSLT [5] and XQuery [4].

Path Query Evaluation

Even though the way in which the document is stored has great impact on how a path query can be evaluated, some common challenges exist in the evaluation of path queries, comparing to their relational peers, and numerous new operators, optimization techniques and index structures have been proposed to facilitate efficient path query evaluation.

One major characteristic of path query is that the query requirements are expressed not only on data entries, such as the tag (label), attributes, and text values of the entries, but also on the structural relationship among the data entries, which are highlighted by the primitives $\uparrow$ (parent axis), $\downarrow$ (child axis), $\uparrow^*$ (ancestor axis), and $\downarrow^*$ (descendant axis).

Navigation is a natural approach for evaluating a path query. The nagivational approach scans the whole document to verify the query requirement on data entries and the structural relationship among data entries. This approach works more naturally while the document is stored as file, in object-oriented database, or in a native data store. In addition, the nagivational approach is potentially very expensive in cost, especially when the ancestor/descendent axis is involved.

The invention of structural join operator [1,13] and multiple algorithms for efficient computing of structural join advanced the evaluation technique of path queries dramatically. A structural join operation takes two sets of data entries as input and returns pairs of data entries that satisfy the desired structural relationship. The basis of the structural join operation is the pre-order and post-order numbering encoding of data entries in the document. The parent-child relationship is a special case of the ancestor-descendant relationship, in which the nodes that satisfies the desired structural relationship have to be exactly one level apart from each other.

$$(a, b) \in \downarrow^*(D) \Leftrightarrow a_{pre} < b_{pre} \wedge a_{post} > b_{post}$$

$$(a, b) \in \downarrow (D) \Leftrightarrow a_{pre} < b_{pre} \wedge a_{post} > b_{post} \wedge a_{level} + 1 = b_{level}$$

The structure join operation and various algorithms that support efficient evaluation of the operation enables the evaluation of path queries by decomposition. This approach partitions a path query into twigs that consists of either a node or a pair of nodes with a desired parent-child or ancestor-descendant relationship. The matching of the nodes can be easily evaluated via indices that are similar to value indices in RDB. The structural relationships between node pairs are evaluated by the structural join. Holistic approach has been proposed and widely adopted in answering path queries. This approach uses a chain of linked stacks to compute and represent intermediate results of a path in a compactly fashion.

Path Query Optimization

As any database management system, optimization is a critical step in evaluating path queries.

Syntax based optimization rewrites a path query into a path query in normal format, in pursuit of minimum expression length, minimum number of predicates, and no redundant value and structural requirements. The rewrite may also aim at decomposing a path query into sub-queries that belong to some fragments of path queries that are simpler to answer.

The cost-based optimization relies on the data statistics of the document to be queried, and the cost-model of the physical operators to be employed, to enumerate various physical evaluation plans and choose one with minimum or acceptable estimate performance. This process involves the study of access method selection, query decomposition, cost estimation, etc.

Indices for Path Query Evaluation

Indexing, being one of the most important ingredients in efficient query evaluation, has seen its importance in the context of XML. Over 20 different types of indices have been proposed and have led to significant

improvements in the performance of XML query evaluation.

Indices similar to the ones used in RDBs, namely value indices on element tags, attribute names and text values, are first used, together with the structural join algorithms [1,3,10,13], in XML query evaluation. This approach turns out to be simple and efficient, but is not capable of capturing the structural containment relationships native to the XML data.

To directly capture the structural information of XML data, a family of structural indices has been introduced. DataGuide [7] was the first to be proposed, followed by the 1-index [11], which is based on the notion of bi-simulation among nodes in an XML document. These indices can be used to evaluate some path expressions accurately without accessing the original data graph. Milo and Suciu [11] also introduced the 2-index and T-index, based on similarity of pairs (vectors) of nodes. Unfortunately, these and other early structural indices tend to be too large for practical use because they typically maintain too fine-grained structural information about the document. To remedy this, Kaushik et al. introduced the $A(k)$-index which uses a notion of bi-similarity on nodes relativized to paths of length k [8]. This captures localized structural information of a document, and can support path expressions of length up to k. Focusing just on local similarity, the $A(k)$-index can be substantially smaller than the 1-index and others. Several works have investigated maintenance and tuning of the $A(k)$ indices. The $D(k)$-index and $M(k)$-index extend the $A(k)$-index to adapt to query workload. The integrated use of structural and value indices has been explored, and there have also been investigations on covering indices and index selection.

Other directions of XML indexing techniques proposed by researchers include indexing frequent subpatterns, indexing XML tree and queries as sequences, forward and backward index, HOPI index, XR-tree, and encoding-based indices.

Key Applications

Path query is the core concept behind the query languages for semi-structured data. It is also the foundation of path algebra that are used to represent and reason about queries expressed against semi-structured data, especially those focusing on retrieving fragments of the document that satisfy certain value and structural constraints.

Future Directions

Even though the basic idea of the path query has been around for decades, its popularity exploded since XML prevails.

Path query has been adopted as the core of expressing queries again semi-structural data, especially XML. Even though XPath query language has been very stable, new language and language features keep emerging. As to the path query itself, there are ongoing study of the sub-languages and the characteristics of such languages, their relationship to each other, the decidability of the queries in these languages, and the complexity of answer the queries.

On the practical side, systems have been developed to answer path queries. Various query evaluation, query optimization and indexing techniques have been proposed to facilitate efficient evaluation of path queries. However, the level of maturity of these techniques are not at the same level as those of relational queries.

In the relational world, the use cases are well understood and various benchmarks have been developed to measure the performance of relational DBMSs. Those of the queries on semi-structured documents are less understood, despite the existence of a few XML benchmarks, such as XMark [12]. In depth research on the usage of path queries and the design and development of benchmarks is yet another promising direction.

Data Sets

Example semi-structured data and queries can be found in benchmarks such as XMark (http://www.xml-benchmark.org), XMach (http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html), and MBench (http://www.eecs.umich.edu/db/mbench).

Cross-references

- ▶ Bi-Similarity
- ▶ Semi-Structured Data Model
- ▶ XML Data Management
- ▶ XPath/XQuery
- ▶ XSL/XSLT

Recommended Reading

1. Al-Khalifa S., Jagadish H.V., Patel J.M., Koudas N., Srivastava D., and Wu Y. Structural joins: a primitive for efficient XML query pattern matching. In Proc. 18th Int. Conf. on Data Engineering, 2002.
2. Benedikt M., Fan W., and Kuper G.M. Structural properties of XPath fragments. Theor. Comput. Sci., 226(1):3–31, 2005.

3. Bruno N., Koudas N., and Srivastava D. Holistic twig joins: optimal XML pattern matching. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 310–321.
4. Chamberlin D., Clark J., Florescu D., Robie J., Simeon J., and Stefanescu M. XQuery 1.0: an XML Query Language, May 2003.
5. Clark J. XSL Transformations (XSLT) version 1.0. http://www.w3.org/TR/XSLT
6. Clark J. and DeRose D. XML Path Language (XPath) version 1.0. http://www.w3.org/TR/XPATH
7. Goldman R. and Widom J. Data Guides: enabling query formulation and optimization in semistructured databases. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 436–445.
8. Kaushik R., Shenoy P., Bohannon P., and Gudes E. Exploiting local similarity for efficient indexing of paths in graph structured data. In Proc. 18th Int. Conf. on Data Engineering, 2002.
9. Koch C. Processing queries on tree-structured data efficiently. In Proc. 25th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems. 2006, pp. 213–224.
10. McHugh J. and Widom J. Query optimization for XML. In Proc. 25th Int. Conf. on Very Large Data Bases, 1999, pp. 315–326.
11. Milo T. and Suciu D. Index structures for path expressions. In Proc. 7th Int. Conf. on Database Theory, 1999, pp. 277–295.
12. 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.
13. Zhang C., Naughton J.F., DeWitt D.J., Luo Q., and Lohman G.M. On supporting containment queries in relational database management systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.

Pattern Based Clustering

► Subspace Clustering Techniques

Pattern Discovery

► Data Mining

Pattern-Growth Methods

Hong Cheng[1], Jiawei Han[2]
[1]Chinese University of Hong Kong, Hong Kong, China
[2]University of Illinois-Urbana-Champaign, Urbana, IL, USA

Definition

Pattern-growth is one of several influential frequent pattern mining methodologies, where a pattern (e.g., an itemset, a subsequence, a subtree, or a substructure) is *frequent* if its occurrence frequency in a database is no less than a specified *minimum_support* threshold. The (frequent) pattern-growth method mines the data set in a divide-and-conquer way: It first derives the set of size-1 frequent patterns, and for each pattern p, it derives p's projected (or conditional) database by data set partitioning and mines the projected database recursively. Since the data set is decomposed progressively into a set of much smaller, pattern-related projected data sets, the pattern-growth method effectively reduces the search space and leads to high efficiency and scalability.

Historical Background

Frequent itemset mining was first introduced as an essential subtask of association rule mining by Agrawal et al. [1]. A candidate set generation-and-test approach, represented by the Apriori algorithm, was proposed by Agrawal and Srikant [2]. The approach effectively reduces the search space by exploring the *downward closure* property of frequent patterns, i.e., *any subpattern of a frequent pattern is frequent*. Various kinds of refinements of and extensions to this approach were proposed afterwards. However, since the Apriori-like candidate set generation-and-test approach repeatedly scans the whole database and checks the candidates by pattern matching, it is still rather costly.

The pattern-growth approach, represented by the *FP-growth* algorithm, was first proposed by Han, Pei, and Yin [8] for mining frequent itemsets. Since then, the method has been developed in several directions: (i) further enhancement of mining efficiency using refined data structures, such as FP-growth* [6], which uses an array-based implementation of prefix-tree-structure of FP-growth; (ii) mining closed and max patterns [6,13], where a pattern p is *closed* if there exists no super-pattern with the same support, and p is a *max pattern* if there exists no super-pattern that is frequent; (iii) mining sequential patterns [11] and frequent substructures [14]; and (iv) mining high-dimensional data set [7] and colossal patterns [15], and pattern-based classification [4] and clustering [12]. A comprehensive overview of such extensions is presented in [7].

Foundations

Frequent Itemset Mining

To illustrate the pattern-growth method, the FP-growth method is briefly introduced here that exploits

pattern-growth in frequent itemset mining. FP-growth works in a divide-and-conquer way. The first scan of the database derives a list of frequent items in which items are ordered by frequency-descending order (Notice that this particular ordering is not essential, and different ordering schemes can be explored). According to this ordering, the database is compressed into a frequent-pattern tree, or *FP-tree*, which retains the itemset association information.

The FP-tree is mined by starting from each frequent length-1 pattern (as an initial suffix pattern), constructing its *conditional pattern base* (a "subdatabase", which consists of the set of prefix paths in the FP-tree co-occurring with the suffix pattern), then constructing its conditional FP-tree, and performing mining recursively on such a tree. The pattern growth is achieved by the concatenation of the suffix pattern with the frequent patterns generated from a conditional FP-tree. Figure 1 shows an example of a global FP-tree as well as a set of conditional trees and the recursive mining process on top of them. Therefore, the FP-growth algorithm transforms the problem of finding long frequent patterns to searching for shorter ones recursively and then concatenating the suffix.

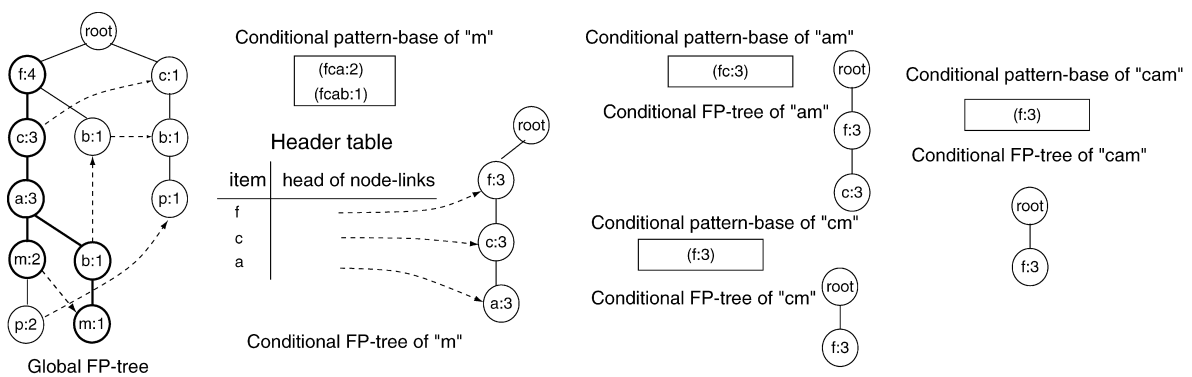
Sequential Pattern Mining

The pattern-growth philosophy has been extended to sequential pattern mining, where a sequential pattern is a set of (gapped) subsequences that occur frequently in a set of sequences. PrefixSpan, developed by Pei et al. [11], is a typical sequential pattern mining algorithm based on the pattern-growth approach. It works in a divide-and-conquer way, by first scanning the sequence database to derive the set of length-1 sequential patterns. Then each sequential pattern is treated as a prefix and the complete set of sequential patterns can be partitioned into different subsets according to different prefixes. To mine the subsets of sequential patterns, corresponding projected databases are constructed and mined recursively.

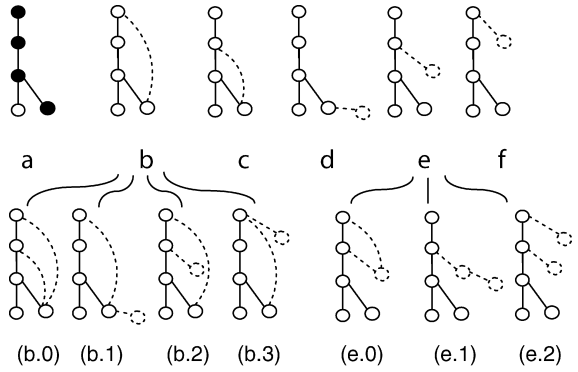
Frequent Subgraph Mining

Inspired by the pattern-growth-based frequent itemset and sequential pattern mining algorithms, there are quite a few pattern-growth-based graph pattern mining algorithms developed. Here gSpan [14] is used an example to explain the ideas.

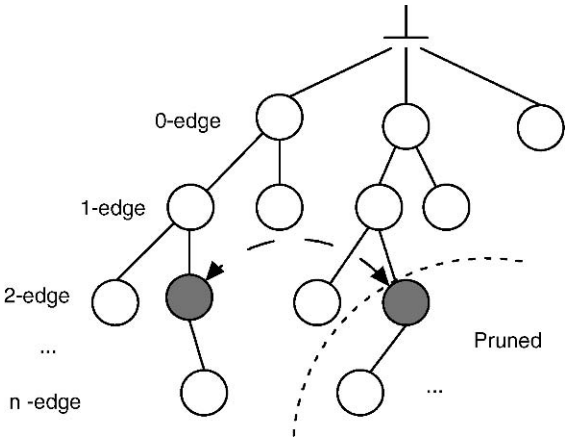
The pattern-growth graph mining algorithm extends a frequent graph by adding a new edge, in every possible position. A potential problem with the edge extension is that the same graph can be discovered many times. gSpan solves this problem by introducing a new lexicographic order among graphs, and maps each graph to a unique minimum DFS code as its canonical label. Based on this lexicographic order, gSpan adopts the pattern-growth philosophy with a *right-most extension* technique, where the only extensions take place on the *right-most path*. A right-most path is the straight path from the starting vertex v_0 to the last vertex v_n, according to a depth-first search on the graph. In Fig. 2, the graph shown in 2a has several potential children with one edge growth, which are shown in 2b–f (assume the darkened vertices constitute the rightmost path). Among them, 2b–d grow from the rightmost vertex while 2e and 2f grow from other vertices on the rightmost path. 2(b.0)–(b.3) are children of 2b, and 2(e.0)–(e.2) are children of 2e. Backward edges can only grow from the rightmost vertex while forward edges can grow from vertices on the rightmost path. The enumeration order of these children is enhanced by the DFS lexicographic order, i.e., it should be in the order of 2b–f.



Pattern-Growth Methods. Figure 1. FP-growth mining on conditional FP-trees.



Pattern-Growth Methods. Figure 2. Graph growth with DFS lexicographic order.



Pattern-Growth Methods. Figure 3. A search space: DFS code tree.

With the DFS lexicographic order definition, the frequent subgraph mining is performed in a *DFS Code Tree*. In a DFS Code Tree, each node represents a DFS code. The relation between parent and child node complies with the parent–child relation; the relation among siblings is consistent with the DFS lexicographic order. Figure 3 shows a DFS Code Tree, the nth level nodes contain DFS codes of $(n-1)$-edge graphs. Through depth-first search of the code tree, all the minimum DFS codes of frequent subgraphs can be discovered. That is, all the frequent subgraphs can be discovered in this way. One should note that if in Fig. 3, the dark nodes contain the same graph but different DFS codes, then one of them must not be the minimum code. Therefore, the search space of that sub-branch can be pruned since it does not correspond to a minimum DFS code.

Key Applications

Pattern-growth methods have been used in many tasks that need the mining of frequent patterns, including the discovery of association and correlation relationships among large sets of transactions, event sequences, or complex structures, the discovery of discriminative frequent features for classification and clustering, as well as many other data mining and pattern recognition applications, such as biological data mining.

Future Directions

Mining Approximate or Noise-Tolerant Patterns

Pattern-growth method has been widely applied for efficient mining of precise and complete set of frequent patterns. However, in some real applications, the capacity to accommodate approximation in the mining process has become critical due to inherent noise and imprecision in complex data sets, for example, gene mutations in genomic DNA sequences, and protein–protein interaction networks. Approximate or noise-tolerant frequent patterns could be the natural choice to handle noise or variations in many applications. Some recent studies proposed new algorithms for mining approximate itemsets [9] and subgraphs [3]. Among them, [9] adopts a candidate generation-and-test approach with the noisy mining model while [3] takes a pattern-growth approach. It is interesting to explore whether pattern-growth method could be naturally applied into a noisy mining model to efficiently discover the approximate or noise-tolerant patterns.

Pattern-Based Classification and Clustering

Frequent patterns have been demonstrated useful in other data mining tasks such as classification [4] and clustering [12], where frequent patterns are used as discriminative classification features and clustering subspaces, respectively. Instead of using the complete set of patterns, usually only a small number of frequent patterns are used in classification and clustering tasks, e.g., the subset of highly discriminative patterns in classification and the subset of frequent patterns with dense regions in clustering. It would be desirable to directly mine the subset of patterns of interests without generating the complete set, for efficiency consideration. To achieve this goal, pruning strategies need to be designed and integrated into the pattern-growth mining methodology to effectively prune the search space which does not yield high-quality patterns. This is a non-trivial task since many quality measures, such as information gain, density, or entropy, are not *anti-monotonic* which is the essential pruning strategy in frequent pattern mining.

Experimental Results

In general, for every proposed method, there is an accompanying experimental evaluation in the corresponding reference. In addition, for frequent itemset mining methods, [5] (The FIMI workshop) provided a detailed and comprehensive experimental evaluation on a large set of benchmark data.

Data Sets

Synthetic Data

A synthetic tree generator can be found at http://www.cs.rpi.edu/~zaki/software

Real Data

A large collection of real transaction datasets can be found at http://fimi.cs.helsinki.fi/data/. Commonly used real graph data includes AIDS anti-viral screening datasets at http://dtp.nci.nih.gov, and NCI anti-cancer screening datasets at http://pubchem.ncbi.nlm.nih.gov.

URL to Code

The binary codes for FP-growth, PrefixSpan and gSpan are provided by the IlliMine project at http://illimine.cs.uiuc.edu/.

The source codes of FP-growth*, FPClose, and FPMax* [6] are provided by Grahne and Zhu at http://fimi.cs.helsinki.fi/src/fimi06.tgz

Cross-references

- ▶ Apriori property and Breadth-First Search Algorithms
- ▶ Frequent Graph Patterns
- ▶ Frequent Itemsets and Association Rules
- ▶ Sequential Patterns

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. and Srikant R. Fast algorithms for mining association rules. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 487–499.
3. Chen C., Yan X., Zhu F., and Han J. gApprox: Mining frequent approximate patterns from a massive network. In Proc. 2007 IEEE Int. Conf. on Data Mining, 2007, pp. 445–450.
4. Cheng H., Yan X., Han J., and Yu P.S. Direct discriminative pattern mining for effective classification. In Proc. 24th Int. Conf. on Data Engineering, 2008.
5. Goethals B. and Zaki M. An introduction to workshop on frequent itemset mining implementations. In Proc. ICDM Int. Workshop on Frequent Itemset Mining Implementations, 2003, pp. 1–13.
6. Grahne G. and Zhu J. Efficiently using prefix-trees in mining frequent itemsets. In Proc. ICDM Int. Workshop on Frequent Itemset Mining Implementations, 2003.
7. Han J., Cheng H., Xin D., and Yan X. Frequent pattern mining: Current status and future directions. Data Mining and Knowledge Discovery, 15:55–86, 2007.
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. Liu J., Paulsen S., Sun X., Wang W., Nobel A., and Prins J. Mining approximate frequent itemsets in the presence of noise: Algorithm and analysis. In Proc. SIAM Int. Conf. on Data Mining, 2006, pp. 405–416.
10. Pan F., Cong G., Tung A.K.H., Yang J., and Zaki M. CARPENTER: Finding closed patterns in long biological datasets. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 637–642.
11. Pei J., Han J., Mortazavi-Asl B., Wang J., Pinto H., Chen Q., Dayal U., and Hsu M.-C. Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Trans. Knowl. Data Eng., 16:1424–1440, 2004.
12. Pei J., Zhang X., Cho M., Wang H., and Yu P.S. Maple: A fast algorithm for maximal pattern-based clustering. In Proc. IEEE Int. Conf. on Data Mining, 2001, pp. 259–266.
13. Wang J., Han J., and Pei J. CLOSET+: Searching for the best strategies for mining frequent closed itemsets. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 236–245.
14. Yan X. and Han J. gSpan: Graph-based substructure pattern mining. In Proc. 2002 IEEE Int. Conf. on Data Mining, 2002, pp. 721–724.
15. Zhu F., Yan X., Han J., Yu P.S., and Cheng H. Mining colossal frequent patterns by core pattern fusion. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 706–715.

PCA

▶ Principal Component Analysis

PDMS

▶ Peer Data Management System

Pedigree

▶ Provenance
▶ Provenance in Scientific Databases

Peer Data Management

► Structured Data in Peer-to-Peer Systems

Peer Data Management System

Philippe Cudré-Mauroux
Massachussetts Institute of Technology, Cambridge, MA, USA

Synonyms

PDMS; Decentralized data integration system

Definition

A Peer Data Management System (PDMS) is a triple $\mathcal{S} = \langle \mathcal{P}, \mathcal{S}, \mathcal{M} \rangle$ where $\mathcal{P}$ is a set of autonomous peers, $\mathcal{S}$ a set of heterogeneous schemas used by the peers to represent their data, and $\mathcal{M}$ a set of schema mappings, each enabling the reformulation of queries between a given pair of schemas.

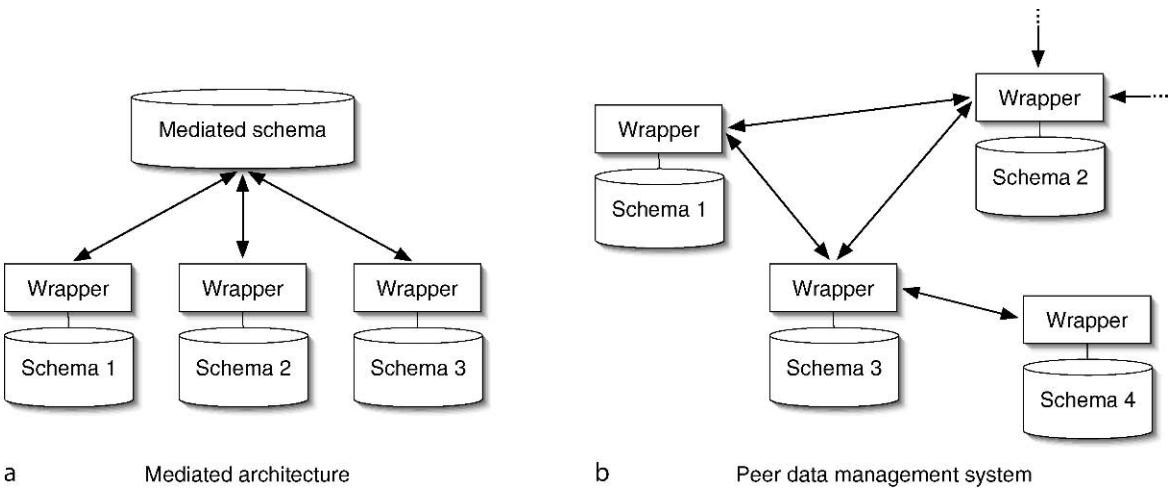
Key Points

A Peer Data Management System (PDMS) is a distributed data integration system providing transparent access to heterogeneous databases without resorting to a centralized logical schema. Instead of imposing a uniform query interface over a mediated schema, PDMSs let the peers define their own schemas and allow for the reformulation of queries through mappings relating pairs of schemas (see Fig. 1). PDMSs typically exploit the schema mappings transitively in order to retrieve results from the entire network.

Compared to centralized data integration systems, PDMSs suggest a scalable, decentralized and easily extensible integration architecture where any peer can contribute data, schemas, and mappings. Peers with new schemas simply need to provide a mapping between their schema and any other schema already used in the system to be part of the network.

The languages used to define the mappings in PDMSs may vary, but are typically derived from GLAV formulae with extensions to support both inclusion and equality mappings. The Piazza system [3] proposes new algorithms to retrieve certain answers in this context. Hyperion [1] is a system focusing on relating data not only at the schema level, but also at the instance level through mapping tables. GridVine [2] provides distributed probabilistic analyses in order to automatically detect mapping inconsistencies in PDMS settings.

PDMSs generally use Peer-to-Peer overlay networks to support their distributed operations. Some use unstructured overlay networks [1,3] to organize the peers into a random graph and use flooding or random walks to contact distant peers. Other PDMSs maintain a decentralized yet structured Peer-to-Peer network [2] to allow any peer to contact any other peer by taking advantage of a distributed index.



Peer Data Management System. Figure 1. Contrary to the mediated architecture (a), Peer Data Management Systems (b) do not impose any form of centralization but consider instead networks of heterogeneous data sources related to each other through pairwise schema mappings.

The lack of global coordination has raised several questions as to the global properties of such systems in the large. In particular, new complex systems perspectives – such as the emergent semantics approach – have been proposed to characterize the global semantics of PDMSs.

Cross-references

- ▶ Data Integration
- ▶ Emergent Semantics
- ▶ Peer-to-Peer Data Integration
- ▶ Peer to Peer Overlay Networks: Structure, Routing and Maintenance

Recommended Reading

1. Arenas M., Kantere V., Kementsietsidis A., Kiringa I., Miller R.J., Mylopoulos J. The hyperion project: from data integration to data coordination. ACM SIGMOD Rec., 32(3):53–58, 2003.
2. Cudré-Mauroux P. Emergent Semantics. EPFL Press, 2008.
3. Halevy A., Ives Z., Madhavan J., Mork P., Suciu D., Tatarinov I. The piazza peer data management system. IEEE Trans. Knowl. Data Eng., 16(7):787–798, 2004.

Peer Database Management

- ▶ Structured Data in Peer-to-Peer Systems

Peer to Peer Network

- ▶ Storage Grid

Peer to Peer Overlay Networks: Structure, Routing and Maintenance

Wojciech Galuba, Sarunas Girdzijauskas
EPFL, Lausanne, Switzerland

Definition

A peer-to-peer overlay network is a computer network built on top of an existing network, usually the Internet. Peer-to-peer overlay networks enable participating peers to find the other peers not by the IP addresses but by the specific logical identifiers known to all peers. Usually, peer-to-peer overlays have the advantage over the traditional client-server systems because of their scalability and lack of single-point-of-failure. Peer-to-peer overlays are commonly used for file sharing and realtime data streaming.

Historical Background

The rise of the Internet brought the first instances of peer-to-peer overlays like the Domain Name System (DNS), the Simple Mail Transfer Protocol (SMTP), USENET and more recently IPv6, which were needed to facilitate the operation of the Internet itself. These peer-to-peer overlays were intrinsically decentralized and represented symmetric nature of the Internet, where every node in the overlay had equal status and assumed cooperative behavior of the participating peers. The beginning of the file-sharing era and the rise and fall of the first file-sharing peer-to-peer system Napster [9] (2000–2001) paved the way for the second generation of peer-to-peer overlays like Gnutella [5] (2000) and Freenet [4] (2001). The simple protocols and unstructured nature made these networks robust and lacking Napster's drawbacks like single-point-of-failure. Since 2001, these peer-to-peer overlays became extensively popular and accounted for the majority of the Internet traffic. Soon after it was evident that the unstructured nature of Gnutella-like systems is embarrassingly wasteful in bandwidth, more efficient structured overlays appeared, like the Distributed Hash Tables (DHTs), which used the existing resources more effectively (e.g., Chord [13]). Currently, unstructured peer-to-peer overlays are sparsely used, as the most popular peer-to-peer applications for file-sharing and data-streaming (e.g., Skype [12], Kademlia [8], KaZaA [6]) are implemented using structured or hybrid overlay concepts.

Foundations

Taxonomy

There are many features of peer-to-peer overlays, by which they can be characterized and classified [2,11]. However, strict classification is not easy since many features have mutual dependencies on each other, making it difficult to identify the distinct overlay characteristics (e.g., overlay topologies versus routing in overlays). Although every peer-to-peer overlay can differ by many parameters, but each of them will have to have certain network structure with distinctive routing

and maintenance algorithms allowing the peer-to-peer application to achieve its purpose. Thus, most commonly, peer-to-peer overlays can be classified by:

1. Purpose of use;
2. Overlay structure;
3. Employed routing mechanisms;
4. Maintenance strategies.

Purpose of Use

Peer-to-peer overlays are used for an efficient and scalable sharing of individual peers' resources among the participating peers. Depending on the type of the resources which are shared, the peer-to-peer overlays can be identified as oriented for:

1. Data-sharing (data storage and retrieval);
2. Bandwidth-sharing (streaming);
3. CPU-sharing (distributed computing).

Data-sharing peer-to-peer overlays can be further categorized by their purpose to perform one or more specific tasks like file-sharing (by-far the most common use of the peer-to-peer overlays), information retrieval (peer-to-peer web search), publish/subscribe services and semantic web applications. The examples of such networks are BitTorrent [3] (file-sharing), YaCy-Peer [14] (web search), etc.

Bandwidth-sharing peer-to-peer overlays to some extent are similar to the data-sharing ones, however, are mainly aimed at the efficient streaming of real-time data over the network. Overlay's ability to find several disjoint paths from source to destination can significantly boost the performance of the data streaming applications. Bandwidth-sharing peer-to-peer overlays are mostly found in peer-to-peer telephony, peer-to-peer video/TV, sensor networks and peer-to-peer publish/subscribe services. Currently Skype [12] is arguably the most prominent peer-to-peer streaming overlay application.

For the computationally intensive tasks, when the CPU resources of a single peer cannot fulfill its needs, a *CPU-sharing* peer-to-peer overlays can provide plenty of CPU resources from the participating idle overlay peers. Currently, only a major scientific experiments employ such strategy for the tasks like simulation of protein folding or analysis of an astronomic radio signals. Although *not* being a pure peer-to-peer overlay, Berkeley Open Infrastructure for Network Computing (BOINC) is very popular among such networks, supporting such distributed computing projects as SETI@home, folding@home, AFRICA@home, etc.

Overlay Structure

Peer-to-peer overlays significantly differ by the topology of the networks which they form. There exist a wide scope of possible overlay instances, ranging from centralized to purely decentralized ones, however, most commonly, three classes of network topology are identified:

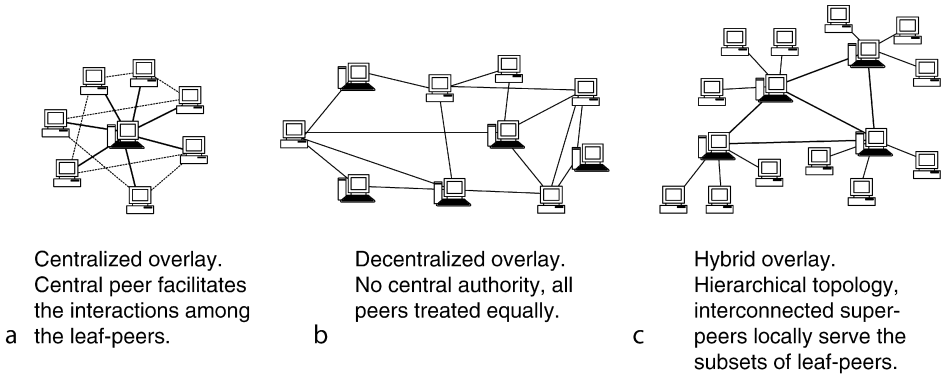
1. Centralized overlays;
2. Decentralized overlays;
3. Hybrid overlays.

Depending on the routing techniques and whether the overlay network was created by some specific rules (deterministically) or in ad hoc fashion (nondeterministically), overlay networks can be also classified into *structured* and *unstructured* peer-to-peer overlays.

Centralized Overlays Peer-to-peer overlays based on *centralized* topologies are pretty efficient since the interaction between peers is facilitated by a central server which stores the global index, deals with the updates in the system, distributes tasks among the peers or quickly responds to the queries and give complete answers to them (Fig. 1(a)). However, not all the purposes of use fit the centralized network overlay model. Centralized overlays usually fail to scale with the increase of the number of participating peers. The centralized component rapidly becomes the performance bottleneck. The existence of a single-point-of-failure (e.g., Napster [9]) also prevents from using centralized overlays for many potential data-sharing applications.

Decentralized Overlays Because of the aforementioned drawbacks, *decentralized* structured and unstructured overlays emerged, which use purely decentralized network model, and do not differ peers as servers or clients, but treat all of them equally – as they were both servers and clients at the same time (Fig. 1(b)). Thus, such peer-to-peer overlays successfully deal with the scalability and can exist without any governing authority.

The simplest decentralized overlays usually are *unstructured*. Unstructured overlay networks typically have arbitrary topology and use flooding based routing among the peers. The distribution of the resources among the peers is completely unrelated to the



Peer to Peer Overlay Networks: Structure, Routing and Maintenance. Figure 1. Examples of peer-to-peer overlays.

network topology. Because of their simplicity, the unstructured overlays are pretty robust to network and peer failures, although are rather inefficient in bandwidth consumption and have poor querying performance.

Structured overlay networks, however, use more efficient routing techniques and the topology of the structured overlays is not arbitrary but typically exhibit Small-World properties, specifically high clusterization and low network diameter. The link establishment among the peers is usually strictly defined by the specific protocols. The topologies can result in various structures like rings, toruses, hypercubes and de-Bruin networks or more loose randomized networks, which do have properties of Small-World networks. A particular instance of structured peer-to-peer overlays is a Distributed Hash Table (DHT) enabling an efficient lookup service, by using a predefined hashing algorithms to assign an ownership for a particular resource (e.g., Chord [13], P-Grid [1], Symphony [7], etc.). In contrast to the traditional Hash Table, the DHTs share the global hashing information among all the participating peers equally and the DHT protocols ensure that any part of a global hash table is easily reachable (usually in logarithmic steps) and there is enough replication to sustain the consistency in the system.

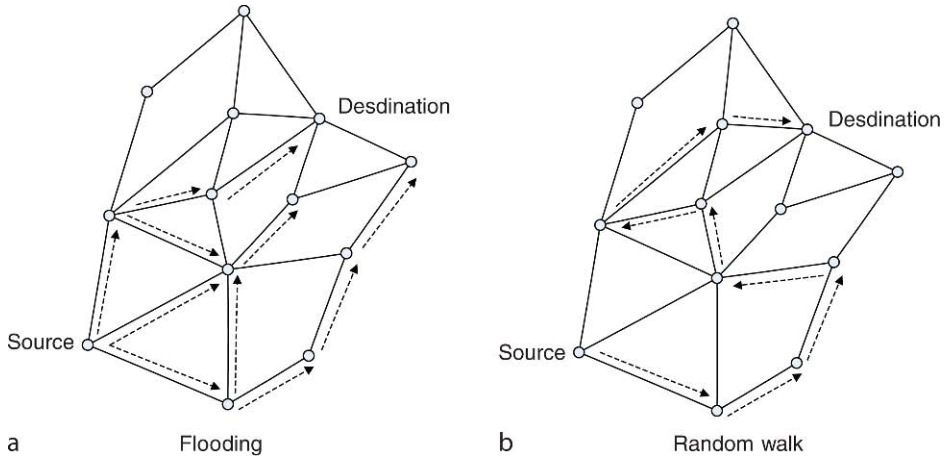
Hybrid Overlays There also exist many *hybrid* peer-to-peer overlays (super-peer systems) which trade-off between different degree of topology centralization and structure flexibility. Hybrid overlays usually use hierarchical network topology consisting of regular peers and super-peers, which act as local servers for the subsets of regular peers (Fig. 1(c)). For example, a hybrid overlay might consist of the super-peers forming a structured network which serves as a backbone for the whole overlay, enabling an efficient communication among the super-peers themselves. Hybrid overlays have advantage over simple centralized networks since the super-peers can be dynamically replaced by regular peers, hence do not constitute single points of failure, but have the benefits of centralized overlays.

Routing

Peer-to-peer overlay networks enable the peers to communicate with one another even if the communicating peers do not know their addresses in the underlying network. For example, in an overlay deployed on the Internet, a peer can communicate with another peer without knowing its IP address. The way it is achieved in the overlays is by *routing* overlay messages. Each overlay message originates at a source and is forwarded by the peers in the overlay until the message reaches one or more destinations. A number of routing schemes have been proposed.

Routing in Unstructured Overlays Unstructured overlay networks use mainly two mechanisms to deliver routed messages: flooding and random walks (Fig. 2). When some peer v receives a flood message from one of its overlay neighbors w, then v forwards the flood message to all of its neighbors except w. When v receives the same flood message again, it is ignored. Eventually the flood reaches all of the destinations.

For example, in Gnutella [10], a file-sharing peer-to-peer system, a peer s that wants to download a file floods the network with queries. If some peer d that has the file desired by s is reached by the query flood, then d sends a response back to s. Flooding consumes a significant amount of network bandwidth. To reduce



Peer to Peer Overlay Networks: Structure, Routing and Maintenance. Figure 2. *Routing in unstructured overlay networks.* The *circles* and *solid lines* represent the overlay topology. The *dashed arrows* illustrate the flow of messages. Peers routing in an unstructured network do not know the exact location of the destinations so they have to either look in all possible directions via flooding or randomly walk to find the destination peer.

it, the flooded messages typically contain a Time-To-Live (TTL) counter included in every message that is decremented whenever the message is forwarded. This limits how far the flood can spread from the source but at the same time lowers the chance of reaching the peer that holds the searched file.

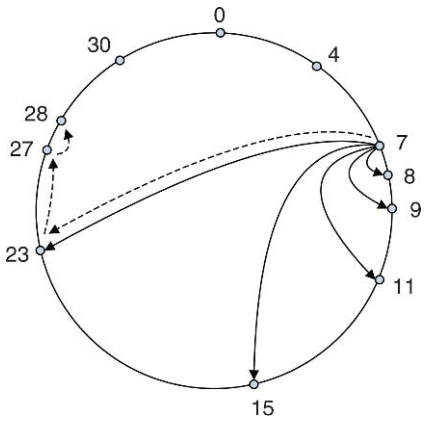
The high bandwidth usage of flooding has led to the design of an alternative routing scheme for unstructured overlay networks: *random walks*. Instead of forwarding a message to all of the neighbors, it is only forwarded to a randomly chosen one. Depending on the network topology random walks provide different guarantees of locating the destination peer(s), however all of the random walk approaches share one disadvantage: a significant and in most cases intolerable delivery latency.

Routing in Structured Overlays As more peers join the overlay network and there are more messages that need to be routed, flooding and random walks quickly reach their scalability limits. This problem has prompted the research on structured overlays.

In structured overlays each peer has a unique and unchanging identifier picked when a peer joins the overlay. The peer identifiers enable efficient routing in the structured overlays. Each routed message has a destination identifier selected from the peer identifier set. Instead of blindly forwarding the message to all neighbors as in the unstructured overlays, a peer in a structured overlay uses the destination identifier to forward the message only to one neighbor. The next hop neighbor is selected to minimize the number of hops to the destination, i.e., the routing is greedy. This selection is made using the *peer identifier distance*.

Most of the modern structured overlays define the notion of distance between any two peer identifiers. For example, in Chord [13] identifiers are selected from the set of integers $[0,2^m - 1]$ and are ordered in a modulo 2^m circle. The distance $d(x, y)$ between two identifiers x and y is defined as the difference between x and y on that identifier circle, i.e., $d(x, y) = (y - x) \bmod 2^m$ In another overlay, Kademlia [8], the identifiers are 160-bit integers and the distance between two identifiers x and y is defined as their exclusive bitwise OR (XOR) interpreted as an integer, i.e., $d(x, y) = x \oplus y$.

Although the modern structured overlays differ in the details of how they make use of the peer identifiers for routing efficiency, they are all based on the same general *greedy routing* principle. When some peer v receives a message with a given destination identifier it forwards the message to that next hop whose identifier is the closest to the destination identifier. In other words, in every hop the message gets as close as possible to the destination. Routing terminates when TTL is exhausted or one of the peers decides it is the destination for the message. The latter decision is application dependent. For example, in a Distributed Hash Table each peer knows for which hash table keys it is responsible. The key space is mapped onto the peer identifier space in DHTs and the destination identifier of each



Peer to Peer Overlay Networks: Structure, Routing and Maintenance. Figure 3. *Routing in chord.* The *big circle* represents the peer identifier space with IDs in the interval $[0,2^5]$. The *small circles* are the peers and the number beside them is their ID. Peer 7 is connected (*solid arrows*) to peers with exponentially increasing distance from 7: $7 + 1 = 8, 7 + 2 = 9, 7 + 4 = 11, 7 + 8 = 15, 7 + 16 = 23$. Assume that peer 7 wants to route a message to peer 28. *Dashed arrows* represent the routing path. The peer 23 is the neighbor of 7 that is closest on the ring to the destination 28 and that neighbor is chosen as the first hop for the message. One hop is not enough to reach 28, but the peer 23 brings the message closer to the destination and peer 27 finally delivers it. The greedy routing rule of always selecting such next hop that brings the message as close as possible to its destination is the main building block of all structured overlay networks.

DHT lookup message specifies the hash table key K the lookup is querying for. The lookup message is greedily routed hop by hop from the origin until the message reaches a peer responsible for K. Routing then terminates and the responsible peer sends a lookup response to the origin. The lookup response contains the hash table value stored under the key K.

Maintenance

Peer-to-peer systems are commonly deployed in environments characterized by high dynamicity, peers can depart or join the system at any time. These continuous joins and departures are commonly referred to as *churn*. Instead of gracefully departing from the network peers can also abruptly fail or the network connection with some of its neighbors may be closed. In all of these cases the changes in the routing tables may adversely affect the performance of the system. The overlay topology needs to be *maintained* to guarantee message delivery and routing efficiency.

There are two main approaches to overlay maintenance: proactive and reactive. In *proactive maintenance* peers periodically update their routing tables such that they satisfy the overlay topology invariants. For example, Chord periodically runs a "stabilization" protocol to ensure that every peer is linked to other peers at exponentially increasing distance. This ensures routing efficiency. To ensure message delivery each Chord peer maintains connections to its immediate predecessor and successor on the Chord ring.

In contrast to proactive maintenance, *reactive maintenance* is triggered immediately after the detection of a peer failure or peer departure. The missing entry in the routing table is replaced with a new one by sending a connect request to an appropriate peer.

Failures and departures of peers are detected in two ways: by probing or through usage. In probe-based failure detection each peer continuously runs a ping-response protocol with each of its neighbors. When ping timeouts occur repeatedly the neighbor is considered to be down and is removed from the routing table. In usage-based failure detection when a message is sent to a neighbor but not acknowledged within a timeout, the neighbor is considered to have failed.

The more neighbors a peer must maintain the higher the bandwidth overhead incurred by the maintenance protocol. In modern structured overlays maintenance bandwidth typically scales as $O(\log(N))$ in terms of the network size.

Key Applications

The key applications of peer-to-peer overlay networks include:

1. File-sharing systems, e.g., BitTorrent [3]
2. VoIP (Voice over IP) and VoD (Video on Demand) systems, e.g., Skype [12]
3. Information retrieval, e.g., YaCy-Peer [14]

Cross-references

- ▶ Load Balancing in Peer-to-Peer Overlay Networks
- ▶ Peer-to-Peer Content Distribution
- ▶ Peer-to-Peer Storage
- ▶ Peer-to-Peer System
- ▶ Peer-to-Peer Web Search

Recommended Reading

1. Aberer K. P-Grid: A self-organizing access structure for P2P information systems. In Proc. Int. Conf. on Cooperative Inf. Syst., 2001.
2. Androutsellis-Theotokis S. and Spinellis D. A survey of peer-to-peer content distribution technologies. ACM Comput. Surv., 36 (4):335–371, December 2004.
3. Bittorrent. http://www.bittorrent.com/.
4. Clarke I., Sandberg O., Wiley B., and Hong T.W. Freenet: A distributed anonymous information storage and retrieval system. In Designing Privacy Enhancing Technologies: Proc. Int. Workshop on Design Issues in Anonymity and Unobservability, 2001.
5. Gnutella Homepage. http://www.gnutella.wego.com/.
6. Kazaa Homepage. http://www.kazaa.com/.
7. Manku G.S., Bawa M., and Raghavan P. Symphony: Distributed hashing in a small world. In Proc. 4th USENIX Symp. on Internet Tech. and Syst., 2003.
8. Maymounkov P. and Mazières D. Kademlia: A peer-to-peer information system based on the XOR metric. In Proc. 1st Int. Workshop Peer-to-Peer Systems, 2002, pp. 53–65.
9. Napster. http://www.napster.com/.
10. Ripeanu M., Foster I., and Iamnitchi A. Mapping the Gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. IEEE Internet Comput. J., 6(1), August 2002.
11. Risson J. and Moors T. Survey of research towards robust peer-to-peer networks: Search methods. Comput. Netw., 50(17):3485–3521, 2006.
12. Skype Homepage. http://www.skype.com/.
13. Stoica I., Morris R., Karger D.R., Kaashoek M.F., and Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. In Proc. ACM Int. Conf. of the on Data Communication, 2001, pp. 149–160.
14. YaCyPeer. http://www.yacyweb.de/.

Peer-To-Peer Content Distribution

Pascal Felber[1], Ernst Biersack[2]
[1]University of Neuchatel, Neuchatel, Switzerland
[2]Eurecom, Sophia Antipolis, France

Synonyms

Cooperative content distribution; Peer-to-peer file sharing

Definition

Peer-to-peer content distribution is an approach for cost-effective distribution of bandwidth-intensive content to large numbers of clients based on *swarming*. Content is split into blocks that are sent to different clients. Thereafter, the clients can directly exchange blocks with one another, in a peer-to-peer manner, without the help of the original server.

Historical Background

Peer-to-peer systems are application-layer networks that directly interconnect users. They have enjoyed a phenomenal success in the last ten years together with the democratization of broadband access. The first disruptive peer-to-peer applications were designed for file sharing (e.g., Napster, Gnutella, KaZaA): they provided users with the means to (i) *search* for content, and (ii) *distribute* content. In the research community, most of the initial focus was on structured peer-to-peer overlays, also known as distributed hash tables (e.g., CAN, Chord). These systems have proved very efficient for *looking up* specific content.

Once users have started exchanging larger files, notably full-length movies, the classical client-server techniques used to transfer files were no longer sufficient and the focus has shifted on developing more scalable mechanisms for content distribution. The first and most emblematic example of peer-to-peer content distribution application is BitTorrent [5]. Its development started in 2001 and it reached widespread popularity in 2003.

BitTorrent is a peer-to-peer application that capitalizes the *bandwidth* of peer nodes to quickly replicate a single large file to a set of clients. The challenge is thus to maximize the speed of replication. The clients involved in a torrent cooperate to replicate the file among each other using *swarming* techniques: the file is broken into fixed size blocks and the clients in a peer set exchange blocks in parallel with one another. BitTorrent only deals with distribution and does not provide any mechanisms for locating content. Several alternatives to BitTorrent have been proposed, but none of them has reached the same level of popularity nor demonstrated sufficient benefits to supersede it.

More recently, some peer-to-peer content distribution systems have been developed specifically for streaming media, such as live television (e.g., PPLive). For such applications, blocks have to be received in order and in time, and clients are only interested in the part of the stream that is being broadcast while they are online.

Foundations

The principle underlying peer-to-peer content distribution is to capitalize the upstream bandwidth of the

clients. A user that downloads some content will, at the same time, send part of the content to other peers.

Peer-to-peer content distribution networks are inherently *self-scalable*, in that the bandwidth capacity of the system increases as more peers arrive: each new peer requests service from, but also provides service to, the other peers. The peer-to-peer system can thus spontaneously adapt to the demand by taking advantage of the resources provided by every peer.

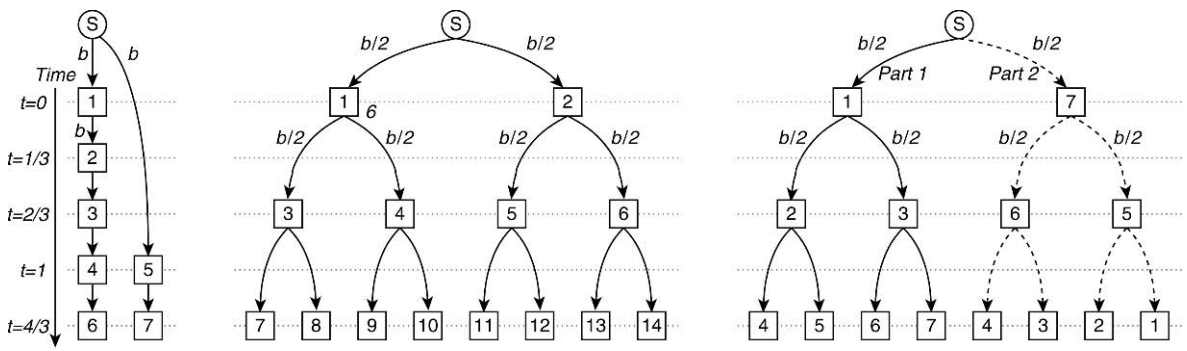
The behavior of each peer in a peer-to-peer content distribution is determined by two factors: (i) the *peer selection strategy* that determines the set of peers a given peer will exchange blocks with and (ii) the *block selection strategy* that determines which blocks will be exchanged. The choice of the peer and the block selection strategy has an important impact on the architecture of the content distribution system. One can distinguish between *structured* and *unstructured* architectures.

Structured architectures organize the peers in a directed acyclic graph such as linear chain or a tree (see Fig. 1). The peer selection is done once for the whole duration of the content distribution. The block selection becomes trivial as each peer simply forwards all the blocks it receives to its children. While structured architectures are easy to understand and model, they are best suitable for content distribution over networks where the bandwidth is homogeneous and nodes stay connected during the whole duration of the content distribution. If these conditions are not met, which is typically the case for the Internet, complex mechanisms are required to make structured approaches work.

Self-organization is the ability of a peer-to-peer network to dynamically determine how to best manage the block exchange as peers join, leave, or fail. Unstructured architectures are self-organizing: they use an underlying mesh structure and build directed graphs through which data is forwarded along several possible paths from the source to each peer. Meshes adapt well to bandwidth fluctuations/heterogeneity and nodes leaving and joining during the transfer at the price of more complex peer and block selection strategies. Since there exist multiple paths for receiving data, each peer must coordinate with its neighbors to avoid receiving the same block multiple times. One such system that uses an unstructured mesh-based approach is BitTorrent, a very popular peer-to-peer system for file distribution.

File Distribution

The BitTorrent protocol makes sensible choices for peer and block selection that are based on a few simple principles. *Peer selection:* first, every peer maintains a limited number of active connections to the other peers that offer the best upload and download rates, thus optimizing bandwidth utilization; second, a peer preferentially sends data to another peer that reciprocally sent data to it, which enforces fairness; third, every peer periodically sends some data to newcomers, so that peers can have an active role in the torrent independent of their arrival time. *Block selection:* a peer requests the block from its neighbors for which the least number of copies exist (rarest first), as rare blocks have a high trading value and can potentially increase the lifetime of the torrent. Maintaining a good



Peer-To-Peer Content Distribution. Figure 1. Evolution of three simple distribution models with time: linear chains (*left*), trees (*center*), and parallel trees (*right*). The nodes in each graph are labeled with the peer number and the edges with the bandwidth allocated for the block transmission. In this example, $C = 3$ and $k = 2$.

diversity of the blocks available in the systems assures that each peer can fully use its upload capacity since it has blocks that its neighbors are interested in.

Despite the simplicity of these design principles and the lack of theoretical foundations to back them when they were introduced, studies [8] have shown they perform exceptionally well in practice. The uplink utilization at the peers is remarkably high, sharing incentives are very effective and resilient to freeriders, and one can observe that peers spontaneously cluster according to their capacity [10].

The choice of the peer and block selection strategy is very important for the good performance of mesh-based systems, because peers can fully use their upload capacity only if the *diversity* of blocks is high, i.e., if they own blocks their neighbors are interested in. Another way to assure that a peer has useful information to transmit to its neighbors is to use *network coding*. In this case, every block that is transmitted is a linear combination of all or a subset of the blocks available at the peer. Since with high probability every such block is unique and contains useful information for the receiving peer, the block selection strategy becomes trivial. Avalanche [6] is a peer-to-peer system for file distribution that uses network coding.

Video Streaming

File distribution systems such as BitTorrent are often used to download files that contain audio or video content. Since the blocks can be downloaded in any order, the consumption of the content cannot start before the download is complete.

In recent years, some peer-to-peer content distribution systems have been developed to support live streaming, where the playback of the content occurs simultaneously with its production. Supporting video streaming is more challenging than file distribution: the users expect that once viewing has started it will be continuous, which means that data must be received in sequence and the rate of data reception must be equal the rate at which data are injected by the source. The architectural variants available for live streaming are the same as for file distribution, namely structured and unstructured approaches.

SplitStream [4] uses a structured architecture and constructs parallel trees, with every peer belonging to all trees. Content is split in multiple layers, each sent along a different tree. As was discussed for the case of file distribution, structured architectures lack flexibility. For this reason, the systems that have been used most widely for video streaming are mesh-based. One can notably mention PPlive [7] that has been used to stream video to tens of thousands of peers.

A robust video streaming system must be able to cope with the heterogeneity of the peers, which can have widely varying download bandwidth. To assure real-time data reception for all clients despite heterogeneity, one can use layered coding: the video is encoded into a base layer and several enhancement layers. While the video cannot be viewed if the base layer is not completely received, the enhancement layer only improve the viewing experience of the video and can be omitted if the download bandwidth is not sufficient.

Many peers are connected via ADSL links that provide much higher download rates than upload rates. A peer-to-peer streaming system is only stable if the aggregate upload rate of the participating peers is at least as high as the aggregate playout rate [9]. In many cases this can only be achieved if there are some peers ("super-peers") that have an upload capacity much higher than the video playout rate [7].

Besides participating in a live streaming event, users may want to watch a video at any point of time, which is referred to as video on demand. While there exist designs for peer-to-peer systems that support video on demand [1], none of these systems has been deployed on a large scale.

Performance Analysis of Distribution Architectures

To evaluate the potential of peer-to-peer file distribution, one can consider very simple distribution models [3] where a server S distributes a file to N peers. The server splits a file into C blocks and serves the file sequentially and infinitely at rate b. The time needed to download the complete file from the server to a *single* peer at rate b is referred to as *one round*.

Consider first a linear chain architecture where the peers are organized in a chain with the server uploading the blocks to the first peer, which in turn uploads the blocks to the second peer and so on (Fig. 1, left). Peers disconnect once they have uploaded the whole file once. The number of peers served in a given number of rounds grows linearly with the number of blocks C, because one can faster engage all peers in the distribution process, and quadratically with the number of rounds t, because the source forks additional chains. Interestingly, when $N/C \ll 1$, the time necessary to serve N peers converges asymptotically to 1.

Consider now a tree architecture where the peers are organized in a tree with an outdegree k (Fig. 1, center). The server serves k peers in parallel, each at rate b/k, and all the peers that are not leaves in the tree in turn upload the blocks to k other peers at rate b/k. This means that it takes k rounds for a peer to download the file and interior nodes upload an amount of data equivalent to k times the size of the file, while leaf nodes do not upload the file at all. The performance of the tree architecture depends on the node degree, and it turns out that for $N/C \leq 1$ the best value is $k = 1$, i.e., a linear chain. For $N/C > 1$ a binary tree is more efficient. Trees with higher degrees are penalized by the higher number of non-contributing leaf nodes.

Finally, consider a forest of k parallel trees, each of which contains all the peers (Fig. 1, right). The server partitions the file into k parts and constructs k spanning trees to distribute each part along a separate tree to all peers. The trees can be built such that each peer is an interior node in at most one tree and a leaf in the remaining $k-1$ trees. The parallel tree architecture is as efficient as the linear chain when $N/C \ll 1$ and significantly outperforms the other two architectures in other scenarios. The optimal performance is obtained for trees with an outdegree $k = e \approx 3$.

Table 1 summarizes the scaling behavior of the different approaches. One can see that for both the tree and parallel trees architectures, the number of clients served increases exponentially in time and in the number of blocks C.

These performance results are remarkable: with the same amount of effort at the server, distributing the content to N peers using a peer-to-peer architecture can be done in a little more than one round, i.e., it takes just slightly longer than to distribute the file from the server to a single peer. While structured architectures are neither robust nor practical, these performance models still give valuable insights on the asymptotically performance of peer-to-peer content distribution. See [11,2] for studies that take into account other factors such as bandwidth heterogeneity.

Developing performance models for unstructured architectures is much more difficult than for structured ones. Yet, it turns out that the following unstructured architecture, called *Interleave*, is analytically tractable [12]. The performance model for Interleave assumes that all blocks are numbered in increasing order, all peers have the same bandwidth, and the transmission of a block takes one slot. Peer selection is random, while block selection is as follows: in every odd time slot the source and the other peers push the highest numbered block they own to a random peer; in every even time slot each peer requests from a random peer the lowest numbered block that the peer does not yet have. Note that Interleave does not require any peer to exchange information with other peers about the block it already has. As a consequence the protocol will experience some inefficiency as a peer p may ask another peer q for a block that q does not have, or may send to q a block that q already has. It can be shown analytically that with high probability the entire file that consists of C blocks can be distributed to all N peers in a time of $3.2 + \frac{\log N}{C}$. Using simulation the result obtained was $2 + \frac{\log N}{C}$, which is similar to the time required by a structured binary tree (see Table 1). Such good performance is quite astonishing as is indicates that there exist robust mesh-based file distribution schemes that can achieve performance as good as structured architectures.

Key Applications

Peer-to-peer content distribution is a technique to transfer large contents simultaneously to many users.

Peer-To-Peer Content Distribution. Table 1. Performance comparison of three structured distribution models

Architecture	Clients served	Service time	Copies served	Download rate	Upload rate
Linear chain	$C \cdot t^2$	$\frac{1}{2} + \sqrt{\frac{1}{4} + \frac{2 \cdot N}{C}}$	1	b	b (except leaves)
Tree	$k^{(t-k)\frac{C}{k}+1}$	$k + (\lfloor \log_k N \rfloor - 1) \cdot \frac{k}{C}$	k	$1/b$	b (except leaves)
Parallel trees	$k^{(t-1)\frac{C}{k}}$	$1 + \lfloor \log_k N \rfloor \cdot \frac{k}{C}$	1	b	b

Key applications include file sharing systems, live TV, video on demand, or distribution of critical updates.

It also represents a cost-effective alternative to scaling up server architectures. It has notably been used as a way to protect servers from unexpected surges in request traffic, called flash crowds, by replicating popular content on multiple peers.

Future Directions

Peer-to-peer content distribution networks are overlays that typically ignore the underlay, i.e., the underlying IP connectivity. Therefore, such networks generate much unnecessary traffic between Internet service providers.

Another important problem that plagues content distribution networks is the fact that the validity of content cannot be verified until after it is downloaded. The media industry contributes to content pollution in existing networks to deter exchange of copyrighted files.

One can expect video on demand to be an important application domain for peer-to-peer content distribution technology. A major challenge is that users must be able to start watching a movie at any time, as well as suspend it, rewind, skip chapters, etc. (VCR functionality). In these scenarios, there is no such synchronization between the peers as with live streaming.

Cross-references

► Distributed Hash Table
► Peer-to-Peer System

Recommended Reading

1. Annapureddy S., Guha S., Gkantsidis C., Gunawardena D., and Rodriguez P. Is high quality VoD feasible using P2P swarming. In Proc. 16th Int. World Wide Web Conference, 2007.
2. Biersack E.W., Carra D., Cigno R.L., Rodriguez P., and Felber P. Overlay architectures for file distribution: Fundamental performance analysis for homogeneous and heterogeneous cases. Computer Networks, 51(3):901–917, 2007.
3. Biersack E., Rodriguez P., and Felber P. Performance analysis of peer-to-peer networks for file distribution. In Proc. 5th International Workshop on Quality of Future Internet Services, 2004, pp. 1–10.
4. Castro M., Druschel P., Kermarrec A.M., Nandi A., Rowstron A., and Singh A. SplitStream: High-bandwidth multicast in a cooperative environment. In Proc. 19th ACM Symp. on Operating System Principles, 2003.
5. Cohen B. Incentives to Build Robustness in BitTorrent. Tech. rep., http://www.bittorrent.org/, 2003.
6. Gkantsidis C., Miller J., and Rodriguez P. Anatomy of a P2P content distribution system with network coding. In Proc. 5th Int. Workshop Peer-to-Peer Systems, 2006.
7. Hei X., Liang C., Liang J., Liu Y., and Ross K. A measurement study of a large-scale P2P IPTV system. IEEE Trans. on Multimedia, 9(8):1672–1687, 2007.
8. Izal M., Urvoy-Keller G., Biersack E., Felber P., Al Hamra A., and Garces-Erice L. Dissecting BitTorrent: Five months in a torrent's lifetime. In Proc. 5th Passive and Active Measurement Workshop, 2004.
9. Kumar R., Liu Y., and Ross K.W. Stochastic fluid theory for P2P streaming systems. In Proc. 26th Annual Joint Conf. of the IEEE Computer and Communications Societies, 2007.
10. Legout A., Liogkas N., Kohler E., and Zhang L. Clustering and sharing incentives in bittorrent systems. In Proc. 2007 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comp. Syst., 2007.
11. Mundinger J., Weber R., and Weiss G. Optimal scheduling of peer-to-peer file Dissemination. Journal of Scheduling, 2007.
12. Sanghavi S., Hajek B. and Massoulie L. Gossiping with multiple messages. IEEE Transactions on Information Theory, 53(12), 2007.

Peer-to-Peer Data Integration

Anastasios Kementsietsidis
IBM T.J. Watson Research Center, Hawthorne, NY, USA

Definition

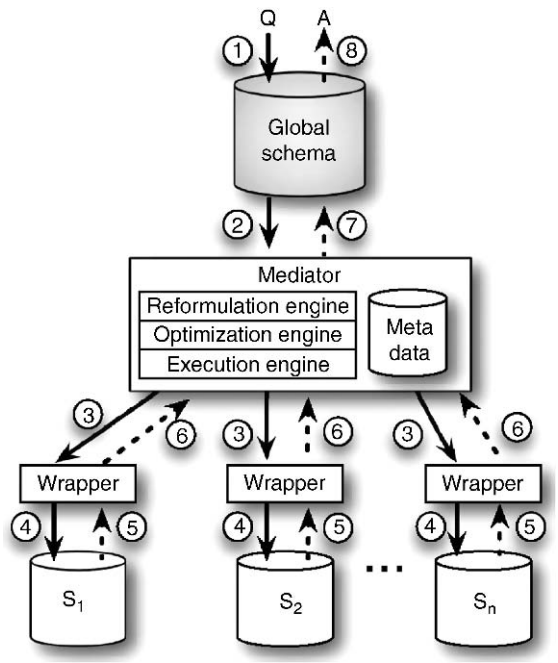
Peer-to-Peer data integration lies in the intersection of two popular research topics, namely, Peer-to-Peer systems and Data Integration, and is one of the key topics in the area of *Peer-to-Peer Data Management*. A Peer-to-Peer data integration setting involves a set $\mathcal{P}$ of autonomous, heterogeneous, independently evolving (peer) sources whose pairwise schema or data-level mappings, collectively denoted by $\mathcal{M}$, induce a peer-to-peer network. In this setting, each (peer) source in the network can be queried and act as an *access point* to the data residing in the other network sources. Research in this area focuses on studying the specification and expressiveness of the peer mappings; the corresponding query languages used; algorithms for rewriting queries between peer source schemas; and, to some extent, topics that concern the propagation of updates between peer sources. Key characteristics of the peer-to-peer data integration setting that differentiate it from *traditional* data integration settings and typical Peer-to-Peer systems include (i) the fact that

each peer can be full-fledged database; (ii) the lack of centralized control and global schemas; (iii) the need for more diverse set of mapping specifications; and (iv) the need for integration of data across diverse domains.

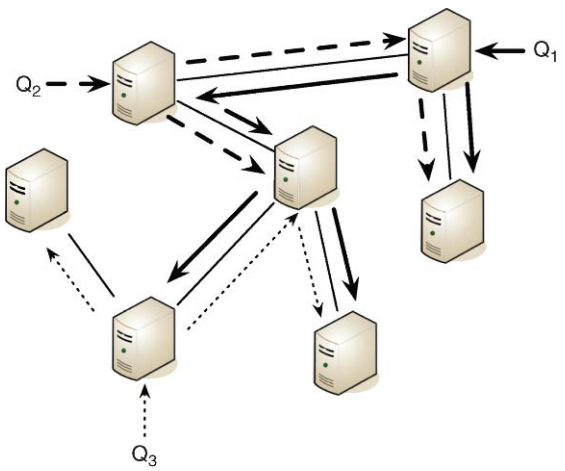
Historical Background

Data integration [11] has been characterized as one of the longest-standing research problems faced by the data management community. A typical data integration setting involves a set $\mathcal{S}$ of sources, a global schema $\mathcal{G}$, and a set $\mathcal{M}$ of mappings between the sources in $\mathcal{S}$ and the global schema $\mathcal{G}$. Figure 1 illustrates such a setting along with the logical steps during query evaluation. Central to the evaluation of queries are the mappings between the global and local schemas (indicated as *Metadata* in the figure). The mappings are used during the rewriting of a user query over the global schema $\mathcal{G}$ to a set of queries over the sources. Two basic approaches have been used to specify the mappings. In a nutshell, in the *Global-as-View* (GAV) approach, the global schema $\mathcal{G}$ is expressed as a view of the set of local schemas $\mathcal{S}$. Main advantage of the GAV approach is the simplicity of the rewriting algorithm. However, a drawback of the approach is that the addition of source in $\mathcal{S}$ results in a revision of the mapping and, in the worst case, a complete re-design of the global schema. In the second approach, called *Local-as-View* (LAV), each source $S_i \in \mathcal{S}$ is expressed as a view over the global schema $\mathcal{G}$. Query rewriting is substantially more involved here, but the approach handles source additions more gracefully.

Peer-to-peer computing involves an open-ended network of computational peers, where each peer exchanges data and services with a set of other peers, called its *acquaintances*. The peer-to-peer paradigm, initially popularized by file-sharing systems such as Napster [12] and Gnutella [5], offers an alternative to traditional architectures found in distributed systems and the web. Distributed systems are rich in services, but require considerable overhead to launch and have a relatively static, controlled architecture. In contrast, the web offers a dynamic, anyone-to-anyone architecture with minimum startup costs but limited services. Combining the advantages of both architectures, peer-to-peer offers an evolving architecture where peers come and go, choose with whom they interact, and enjoy some traditional distributed services with less startup cost. Figure 2 shows an example of such a system, often referred to in the literature as an *unstructured* peer-to-peer system, to differentiate it from the complementary class of *structured* peer-to-peer systems whose architecture is based on *Distributed Hash Tables* (DHT's). As shown in the figure, mappings in unstructured peer-to-peer system (represented by solid lines) exist between any pair of peers while queries (represented by various line arrows) can be initiated



Peer-to-Peer Data Integration. Figure 1. Typical data integration setting.



Peer-to-Peer Data Integration. Figure 2. A Peer-to-Peer system.

at any peer in the system. Furthermore, different queries might involve a different set of peers.

Probably the first works to consider the interaction between database and peer-to-peer systems were the ones by Gribble et al. [6] and Bernstein et al. [3]. The former work focuses on the problem of *data placement*, that is, to find an optimal placement of a set of objects on a peer-to-peer system, under a pre-specified query workload, so as to minimize the cost of evaluating the queries. In more detail, the authors consider graph whose nodes correspond to the peers in a peer-to-peer network. Each node is associated with a storage capacity and a query workload. Furthermore, each pair of nodes connected by an edge has an associated data trasfer cost, over this edge. Queries over this network are object lookups and each query has an associated frequency and cost (with the cost being zero, if the query can be served locally in a peer, and a function of the object size and edge transfer cost, if served remotely). In this setting, the main result of this work is that the problem of data placement with optimal cost is NP-complete. The work by Gribble et al. [6] initiated the Piazza System [17] (described in the next section) although the follow-up work in Piazza addresses a setting much closer to data integration. As presented here, the setting in Gribble et al. [6] is more closely related to the one found on the work on DHT's like CAN [15] or Chord [16].

Bernstein et al. [3] is the first work to actually discuss data integration in a peer-to-peer context. The focus in this work is the introduction of the Local Relational Mode (LRM), a model designed specifically for the peer-to-peer setting. Key notions in the model are that of *coordination formulas* and *domain relations*. Much like a mapping in a typical data integration setting, a coordination formula establishes a relationship between the data items stored in acquainted peers. This relationship can be used to express a constraint, like for example, that a particular data item must be stored in either one of three different acquainted peer databases, or it can be used during query answering by expressing, for example, a GAV (LAV) type mapping where the data items returned by query Q_1 in a peer P_1 are contained in those returned by query Q_2 in peer P_2. Since different peers might use different vocabularies to express the same real-world notion, it is the responsibility of domain relations to express the mapping between these vocabularies. For example, a domain relation might be used to map prices expressed in US dollars, in a peer, to those in euro in another peer. The work by Bernstein et al. [3] initiated the Hyperion project [2] the details of which are described in the next section.

Foundations

Important aspects in every peer-to-peer data integration system include (i) the peer joining process and especially the type of mappings used by the system to resolve the heterogeneity of the acquainted peer; (ii) the supported query language and the processing of queries; and (iii) how the system deals with the lack of centralization and the inherent dynamic and unreliable nature of the peer-to-peer setting. Not every system addresses all these aspects and different systems put more emphasis on a different system aspect. The following paragraphs provide an overview of some of the main systems in this area.

In Piazza [17], each peer in the system *exports* a schema which can be one of two kinds: (i) a virtual schema, used for querying and mapping the schemas of its acquainted peers, or (ii) a schema which internally is mapped to actual stored data in the peer. In either case, a peer joining the system establishes GAV or LAV mappings between its export schema and the export schemas of peers that are already part of the system. The creation of such mappings is a complicated process which cannot be fully automated. *Schema matching* techniques are used, both in Piazza and elsewhere, to facilitate their creation. Both the created mappings and the supported query language are expressed in a fragment of XQuery. User queries in this fragment are expressed over the schema of a single peer. The query is answered locally by the peer, if the peer actually stores data, and it is also reformulated to a set of queries over the acquaintances of the current peer. Briefly, the reformulation algorithm combines view unfolding, if a GAV mapping exists between acquainted peers, and an answering queries using views algorithm [7], in the case of LAV mappings. The reformulation algorithm terminates once all the queries that result in from the reformulation refer only to relations that correspond to stored data in peers (and no virtual schemas).

Influenced by the work in the LRM model, in Hyperion [2] two types of mappings are used to support the sharing of information between relational database peers, namely, *mapping expressions* and

mapping tables. Similar to the GAV/LAV mappings in Piazza (and elsewhere), mapping expressions are schema-level mappings used during query answering for the reformulation of queries between peers. A distinguishing feature of Hyperion, mapping tables [9] are *data-level* mappings that associate the values of acquainted peers (inspired by the domain relations in the LRM model). A mapping table T is a relation over the set of attributes $X \cup Y$, where X and Y are non-empty sets of attributes belonging to two acquainted peers. Each tuple $t \in T$ (whose values might include constants and/or variables) associates the set of values in $t[X]$ to those in $t[Y]$.

While schema matching can still be employed for the creation of mapping expressions, the creation of mapping tables requires the development of specialized techniques. Indeed, the work in [9] formalizes mapping tables as data-level constraints over the sharing of data between peers and illustrates how new mapping tables can be inferred automatically (from existing tables) while establishing an acquaintance between two peers. Due to the dynamic nature of peer-to-peer system, peers are expected to continuously evolve and change their schema (less frequently) and their data (very frequently). This change influences both the existing mapping expressions and the existing mapping tables. As a result, work in the Hyperion project is concerned with how to maintain the existing mappings with emphasis in the frequently updated mapping tables.

In terms of query answering, for the case of mapping expressions, Hyperion relies on techniques similar to the ones developed for Piazza. For the case of mapping tables, Hyperion uses a specialized algorithm to rewrite select-project-join (SPJ) queries between peers, relying only on the use of mapping tables for the rewriting [10]. No algorithm is provided in Hyperion to rewrite queries by combining mapping expressions and tables. However, mapping expressions are used in conjunction with mapping tables in Hyperion for *data coordination* [8]. Indeed, another distinguishing characteristic of Hyperion is that it supports the creation and distributed execution of Event-Condition-Action (ECA) rules over multiple acquainted peer.

The PeerDB [14] system is built on top of a generic peer-to-peer platform, called BestPeer [13]. The BestPeer platform supports two types of peers, namely, a large number of normal (data) peers and a smaller number of *location independent global names lookup (LIGLO)* peers. A LIGLO peer acts as a name server with which every peer in the system registers, when joining, in order to acquire a unique identifier.

Similar to Piazza and Hyperion, in PeerDB, data sharing between heterogeneous peers is achieved in two steps. In the first step, mappings are established between the schemas of the peers. In the second step, the mappings are used to rewrite a query over the schema of one peer to a query over the schema of another. PeerDB has a number of distinguishing characteristics over the previous approaches. First, it differs from the previous systems in that it uses an information retrieval-based technique as a basis for its schema mappings. In more detail, each peer relation and attribute name is associated with a set of descriptive keywords. A mapping between two relations (attributes) that reside on different peers is established if their corresponding descriptive keywords overlap *significantly*. Once a mapping is established, it is used to rewrite a query that refers to one relation into a query that refers to its mapped counterpart. The second distinguishing characteristic of PeerDB is that mappings are established dynamically, at query time. In more detail, PeerDB employs an agent-based technology both during the discovery of schema mappings and during the rewriting of queries. After a user initiates a query over a local peer schema, software agents are responsible for crawling the peer-to-peer network, looking for peers whose schemas can be mapped to the schema where the user query is posed. The agents carry all the necessary functionality to perform the rewriting once such schemas are discovered.

Similar to PeerDB, the GridVine [4] system is built on top its own generic peer-to-peer platform, called P-Grid [1]. However, while the architecture of BestPeer follows the peer/super-peer paradigm, the architecture of P-Grid is that of a structured overlay network. Capitalizing on the efficiency of P-Grid in terms of indexing, routing and load balancing, GridVine builds a semantic mediation layer on top of P-Grid. In this semantic layer, schemas and data are represented as RDF triples, while queries over these triples are expressed as *triple patterns*. A triple pattern query issued at any peer is routed to the peer that can answer the query by using the services of the overlay network (by hashing the constants appearing in the query). In terms of heterogeneity, GridVine employs OWL

statements to relate semantically similar schema elements that belong to pairs of schemas. During query evaluation, these mappings are used for the rewriting of queries which are then executed using the overlay network, as described earlier.

Key Applications

Scientific databases, Health informatics databases, Business-to-Business (B2B).

Cross-references

- ▶ Data Integration
- ▶ Distributed Database Systems
- ▶ Parallel Database Systems
- ▶ Peer Data Exchange
- ▶ Peer-to-Peer Data Management
- ▶ Schema Matching

Recommended Reading

1. Aberer K., Cudré-Mauroux P., Datta A., Despotovic Z., Hauswirth M., Punceva M., and Schmidt R. P-Grid: a self-organizing structured P2P system. ACM SIGMOD Rec., 32(3):29–33, 2003.
2. Arenas M., Kantere V., Kementsietsidis A., Kiringa I., Miller R.J., and Mylopoulos J. The hyperion project: from data integration to data coordination. ACM SIGMOD Rec., 32(3):53–58, 2003.
3. Bernstein P., Giunchiglia F., Kementsietsidis A., Mylopoulos J., Serafini L., and Zaihrayeu I. Data management for peer-to-peer computing: a vision. In Proc. 5th Int. Workshop on the World Wide Web and Databases, 2002.
4. Cudre-Mauroux P., Agarwal S., and Aberer K. GridVine: an infrastructure for peer information management. IEEE Internet Comput., 11(5):36–44, 2007.
5. Gnutella Protocol Specification. World Wide Web URL: http://gnet-specs.gnufu.net/.
6. Gribble S., Halevy A., Ives Z., Rodrig M., and Suciu D. What can databases do for peer-to-peer? In Proc. 4th Int. Workshop on the World Wide Web and Databases, 2001.
7. Halevy A.Y. Answering queries using views: a survey. VLDB J., 10(4):270–294, 2001.
8. Kantere V., Kiringa I., Mylopoulos J., Kementsietsidis A., and Arenas M. Coordinating peer databases using ECA rules. In Proc. Int. Workshop on Databases, Information Systems and Peer-to-Peer Computing, 2003, pp. 108–122.
9. Kementsietsidis A., Arenas M., and Miller R.J. Data mapping in peer-to-peer systems: semantics and algorithmic issues. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 325–336.
10. Kementsietsidis A. and Arenas M. Data sharing through query translation in autonomous sources. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 468–479.
11. Lenzerini M. Data integration: a theoretical perspective. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2002, pp. 233–246.
12. Napster. World Wide Web URL: http://www.napster.com/.
13. Ng W.S., Ooi B.C., and Tan K.-L. BestPeer: a self-configurable peer-to-peer system. In Proc. 18th Int. Conf. on Data Engineering, 2002, p. 272.
14. Ng W.S., Ooi B.C., Tan K.-L., and Zhou A. PeerDB: a P2P-based system for distributed data sharing. In Proc. 19th Int. Conf. on Data Engineering, 2003, pp. 633–644.
15. Ratnasamy S., Francis P., Handley M., Karp R., and Shenker S. A scalable content addressable network. In Proc. ACM Int. Conf. on Data Communication, 2001, pp. 161–172.
16. Stoica I., Morris R., Karger D., Kaashoek F., and Balakrishnan H. Chord: a scalable peer-to-peer lookup service for internet applications. In Proc. ACM Int. Conf. on Data Communication, 2001, pp. 149–160.
17. Tatarinov I., Ives Z., Madhavan J., Halevy A., Suciu D., Dalvi N., Dong X.L., Kadiyska Y., Miklau G., and Mork P. The piazza peer data management project. ACM SIGMOD Rec., 32(3):47–52, 2003.

Peer-to-peer Database

- ▶ Distributed Architecture

Peer-to-peer File Sharing

- ▶ Peer-To-Peer Content Distribution

Peer-to-peer Network

- ▶ Peer-to-Peer System

Peer-to-peer Overlay

- ▶ Peer-to-Peer System

Peer-to-Peer Publish-Subscribe Systems

Peter Triantafillou, Ioannis Aekaterinidis
University of Patras, Rio Patras, Greece

Definition

Publish/Subscribe (a.k.a. pub/sub) software systems constitute a facility for asynchronous filtering of

information. Users, consumers of information, present the system with continuous queries, coined *subscriptions*. Sources of data generation (producers) present the system with data-carrying *publication events*. The pub/sub system infrastructure is responsible for (asynchronously) matching the publication events to all relevant subscriptions. Hence, in essence, this infrastructure filters all available information for every user and presents to each user only the information units (s)he has defined as relevant. As such, a pub/sub infrastructure can play a vital role in large-scale data systems, with huge volumes of data, shielding users from the burden of always actively searching for and retrieving relevant information units.

Peer-to-Peer (P2P) systems are software systems, which in fact constitute *overlay* networks, which are built over physical networks, such as the Internet. Their key discriminative feature is the complete decentralized algorithmic and system design, which, in turn, leads to guarantees with respect to system *scalability* in terms of the network size and stored data. In addition, P2P systems are characterized by *self-organization*, being able to withstand high dynamics with respect to network nodes joining and leaving the system while continuing to offer efficient operation.

Pub/sub P2P systems are an attempt to combine these two important technologies. The central aim is to offer facilities for asynchronous matching of continuous user queries to publication events, at very large scales. This entails dealing with large numbers of producers and consumers of information, which are geographically distributed and, also, supporting large publication-event and subscription arrival rates. Coupling pub/sub technology with the P2P paradigm achieves this aim, while introducing the additional benefits of scalability and self-organization into the pub/sub realm.

Historical Background

Publish/subscribe systems have evolved significantly over time, differing on a number of fundamental characteristics. They are classified into two major categories, according to the way subscribers express their interests; the *topic-based* [6,10,16] and the *content-based* [4,5,7,8] systems. Historically, the first systems were topic-based, in which users subscribe to specific topics. All incoming publication events associated with a particular topic are sent to all user subscribers of the topic. This paradigm mimics the way news groups operate. *Content-based* pub/sub systems emerged subsequently, offering users the much-needed ability to express their interests on specific publication events, carrying specific content. Principally, users issue subscriptions which specify predicates over the *values* of a number of well defined attributes. The matching of publication events to subscriptions is performed based on the content (i.e., the values of attributes) being carried by the publication events.

At the turn of the century, related research efforts were maturing and a number of content-based pub/sub systems were already available. Influential examples, with respect to content-based pub/sub systems research, include SIENA [5], Gryphon [4], Le Subscribe [8] and JEDI, [7]. Already, some of this work targeted the challenging issues arising in a distributed system, where publishers and subscribers are geographically distributed [4,5]. Around the same time and in parallel, research related to P2P networks and systems was also maturing. Worthy of special mention are *structured P2P overlay networks*, like Chord [14], Pastry [13], CAN [12], and P-Grid [1]. Several of these networks influenced work on pub/sub systems, especially endeavors aiming to leverage existing P2P networks for providing pub/sub functionality with scalability, efficiency, and self-organization. The Scribe system [6] was a first attempt at providing topic-based pub/sub functionality over the Pastry P2P network. This effort was followed by endeavors to build content-based publish/subscribe systems over P2P networks, [2,3,9,11,15,17].

Foundations

Pub/sub data management represents a significant point of departure compared to traditional data management. In the latter, data items are stored and the system is responsible for appropriate indexing and processing queries and updates issued by users against these data items. In the pub/sub model, it is the users' continuous queries (subscriptions) that are stored and indexed appropriately and the system is responsible for processing data-carrying publication events, matching them to all relevant stored subscriptions.

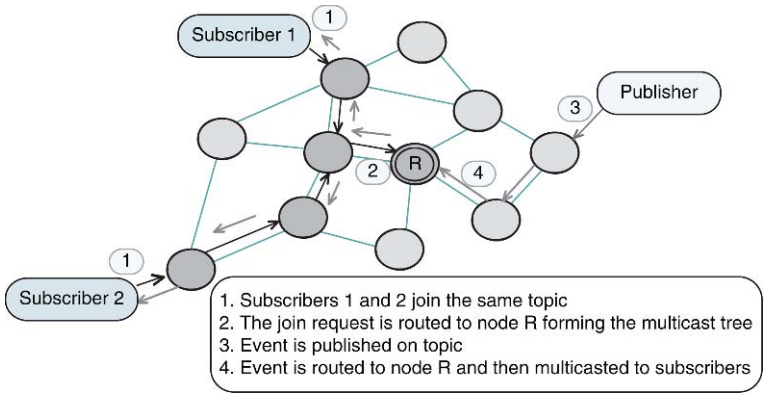
The fundamental functionality offered by a publish/subscribe system rests on the pillars of subscription processing and publication-event processing and matching. In a P2P environment, all this functionality is based on distributed algorithms that appropriately leverage the P2P network capabilities in order to ensure scalability and efficiency of operation, free of concern

for network topology dynamics. The core functionality exported by a P2P network is the so-called *lookup()* function, which receives as input a *key* and returns the network address of the peer node where the data item associated with the key is located. Structured P2P networks, such as those built using *Distributed Hash Tables (DHTs)* (such as [12–14]) can ensure that *lookup()* executes in $O(logN)$ messages, in a network of N nodes, a feature that in essence guarantees scalability and efficiency.

In topic-based publish/subscribe systems, events and subscriptions are associated with specific topics (a.k.a. groups/subjects). A straightforward approach, on which Scribe [6] for example is based for supporting topic-based pub/sub functionality in a P2P environment is to further associate each topic with a *multicast tree*. Topic (and thus multicast tree) creation, involves identifying the node responsible for the tree root, which is determined simply by issuing a *lookup (hash(topic_name))*, using the DHT's hash function. The DHT hash function introduces a randomizing effect when selecting root nodes for different multicast trees. Subscription processing then involves storing the subscription locally at the origin peer node and having that node join the multicast tree. This is basically accomplished by also issuing a *lookup(hash(topic_name))* and creating a path in the multicast tree consisting of all the DHT nodes visited during the execution of the *lookup()* call. Finally, publication-event processing is performed by locating the tree root node (again, using the *lookup ()* function) and sending the publication to it, which it subsequently distributes to the multicast tree. Figure 1 illustrates this process.

The content-based pub/sub model has dominated the area, since it allows for greater user-query expressiveness, resulting in more efficient information filtering. However, this model incurs a much higher complexity. The publication event and subscription schema adopted in this model, defines a set of A attributes (a_i, $1 \leq i \leq A$). Each attribute a_i consists of a name, a type (usually string or numerical), and a value $v(a_i)$. A publication event is defined to be a set of $k <$ *attribute,value* $>$ pairs ($k \leq A$), while a subscription is defined through an appropriate set of predicates on attributes' values over a subset of the A attributes of the schema. The complexity emerges from the need to support subscriptions with complex predicates. For numerical-typed attributes, the allowable predicates may involve equality, inequality, $\leq$,$\geq$, and ranges of values. For string-typed attributes, subscribers may define prefix, suffix, sub-string and equality predicates. Fundamentally, solutions in a P2P environment can be classified according to whether they require knowledge of the internal DHT routing state, which is maintained by each DHT node, and its association with additional state that is needed for subscription and publication processing [3,11,15]. A key characteristic in several of these approaches [11,15] is that each node n_1 associates with each other node in its routing table, say n_2, additional state that consists of all subscriptions that n_1 has received from n_2. When a publication event arrives at n_1, it is forwarded to every node n_2 in its routing table only if the publication event matches one of the subscriptions sent to n_1 by n_2.

Approaches that do not belong in this category avoid the maintenance costs associated with the extra,



Peer-to-Peer Publish-Subscribe Systems. Figure 1. Multicast tree construction for event dissemination in topic-based publish/subscribe.

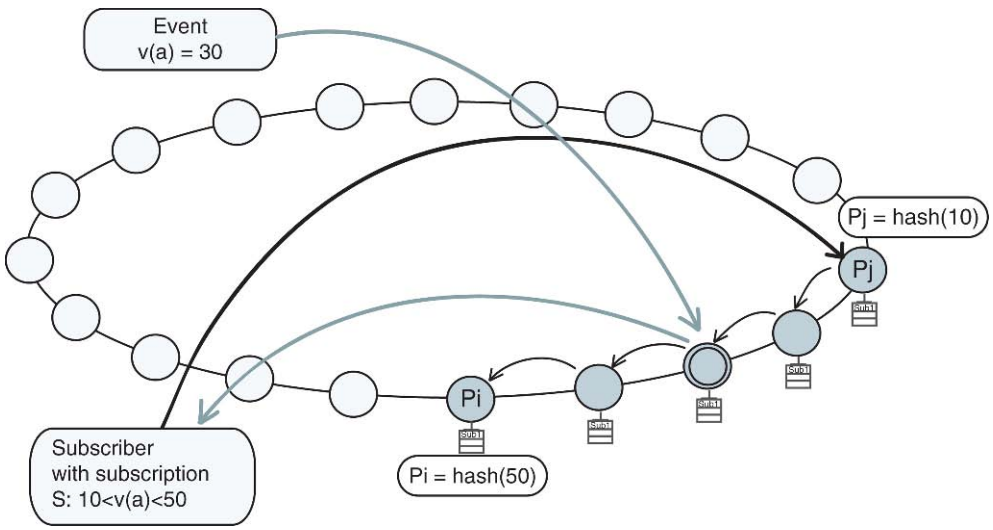
per-node state and enjoy wider applicability as they can be easily integrated with a number of DHTs. Hereafter, the focus is on these approaches, where the content (i.e., the values defined by the predicates) will determine at which peer nodes the subscriptions will be stored. Similarly, the values carried by a publication event will determine which route must be followed within the network so to reach every possible peer node storing a relevant subscription.

Equality predicates can be handled straightforwardly – the basic idea being the following. A subscription s associating with attribute a the value $v(a)$, can be stored simply at the peer node with identifier p, where $p = hash(v(a))$. A publication event e carrying $a = v(a)$, is processed by visiting the peer node $p = hash(v(a))$ and retrieving all locally-stored subscriptions, such as s.

However, for more complex predicates, this is inadequate. At an abstract, high-level view, proposed solutions, albeit very different, share the following characteristics. The node ID namespace is typically associated in solution-specific manners with the value domain of an attribute. A subscription with a complex predicate on an attribute is stored at several nodes, whose IDs depend on the values defined by the subscription's predicate. In other words, a subscription is mapped to a subspace of the node ID namespace. Publications, associating an attribute with a value correspond to a point in the namespace. The node associated with this point, by construction, will be storing all subscriptions whose predicates on this attribute include this event's value. In this way, the *subscription to publication event rendezvous* occurs and in this rendezvous node the event can be matched to all relevant subscriptions.

For concreteness, the basics of the approach adopted in [17] – one of the first to leverage an existing P2P network on top of which to build scalable content-based pub/sub systems – for processing subscriptions with range predicates, is outlined. The approach is simple, requires no additional state maintenance, and no knowledge of the internals of the underlying DHT state. This approach utilizes the Chord DHT [14] in which nodes are arranged in a circular list according to their IDs. However, in [17] specific order-preserving hashing is employed to store subscriptions in the Chord network. That is, when processing subscriptions with values $v(a)_i$ and $v(a)_j$, they are stored at nodes p_i and p_j whose IDs are given by $hash(v(a)_i)$ and $hash(v(a)_j)$, respectively. If $v(a)_j < v(a)_i$, then $p_j < p_i$. An incoming subscription s, identifying a range of values $[v(a)_j, v(a)_i]$ is stored on all nodes in the arc of the ring starting at node $hash(v(a)_j)$ and ending at node $hash(v(a)_i)$. Given this, a publication carrying value $v(a)_k$, with $v(a)_j < v(a)_k < v(a)_i$ will be directed at node $hash(v(a)_k)$ and this node by construction falls on the arc of the ring where the subscription s has been stored. Thus, the matching can be locally performed at this node. Figure 2 illustrates this process.

Finally, the aforementioned discussion has focused on single-attribute (one-dimensional) subscriptions and publications. In general, pub/sub systems involve subscriptions and publications specifying multi-dimensional



Peer-to-Peer Publish-Subscribe Systems. Figure 2. Processing range queries in a content-based pub/sub system.

content. Processing multi-dimensional publication events and subscriptions is a source of additional complexity. In this setting, a publication event *e* will match a subscription *s* if and only if all attributes named by *s* are also named by *e* and the predicates defined by *s* on its attributes are satisfied by the values for these attributes carried by the publication event. A straightforward approach dealing with multi-dimensional events and subscriptions is the following. First, perform the processing as discussed above for each named attribute in the publication event and subscription. Second, for each attribute collect a candidate result set, consisting of the subscriptions being matched only for that attribute of the publication. Lastly, merge and filter the per-attribute candidate result sets into a single result set, by removing those subscriptions in the candidate result sets which have at least one attribute predicate not satisfied by the publication event. Figure 3 illustrates this process.

As these candidate result sets are distributed and can contain very large numbers of subscriptions, the tasks of merging them and filtering subscriptions from them can introduce large overheads. Therefore, multi-dimensional event processing is open to a number of crucial performance optimizations which reveal key trade-offs with respect to network bandwidth overheads, number of required messages, and event-matching latencies [2].

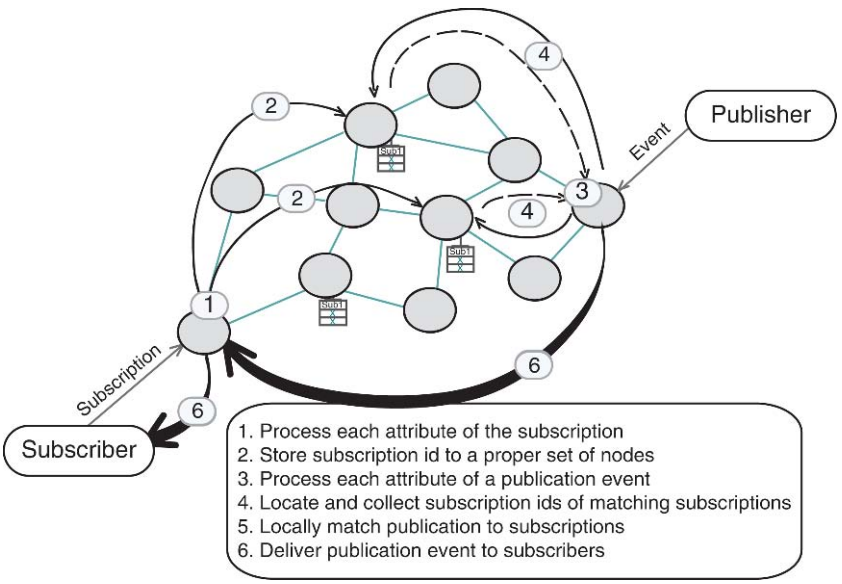
Key Applications

In recent years one notices a proliferation of data-intensive and compute-intensive networked applications aiming for efficiency, scalability, and self-organization. A key characteristic in many such applications is the massive amounts of data of various types and characteristics being generated continuously, from many different sources, at different times, and possibly at very high rates. Users can thus be inundated by the sheer volume of this data and are confronted with severe difficulties in accessing only the typically very small fraction of this data that is of interest to them. Hence, what is very much needed for such applications is an infrastructure that can offer decentralized, scalable, self-organizing asynchronous filtering of this massive information base.

Content-based publish subscribe systems operating in the distributed environment of a P2P overlay network appear as the appropriate infrastructure, and are used to design and implement such applications. A few representative example applications, which also indicate open research and development challenges, are listed below:

(i) *Information Feeds*

A classic example for publish/subscribe application infrastructures are data applications based on information feeds. In these applications publishers can be, for example, news agencies that publish



Peer-to-Peer Publish-Subscribe Systems. Figure 3. Multi-dimensional event and subscription processing in content-based P2P publish/subscribe systems.

news articles. Human subscribers declare their interests with proper content-based subscriptions involving numerical-attribute predicates such as date ranges, and string-attribute predicates such as words defining article title prefixes. In general, any RSS feed-like information dissemination system like stock exchange reports falls in this category. The distributed aspects occur when considering large, multi-national news agencies whose computers form a large, geographically-dispersed network. One can even further imagine alliances of such multi-nationals, increasing dramatically the application's scale.

(ii) *Grid Computing*

Consider a farm of computing clusters, each one including a number of computing nodes with heterogeneous characteristics involving, for example, various types of operating systems, hardware specifications, etc. and forming a peer-to-peer computing grid. Each node publishes its characteristics while possible users that wish to execute their programs are interested in specific node attributes (subscriptions) defining for example acceptable CPU speed ranges, minimum memory requirements, desirable operating system versions, etc.

(iii) *Pub/sub and the Web*

The web is of course a massive distributed data repository. Web information systems can be significantly enriched by adopting pub/sub technology. Overlay networks can be created consisting of nodes representing web pages. A pub/sub system over this overlay network can be used, for instance, to inform existing web users of interesting new pages being created, of updates to pages already marked as relevant, etc.

(iv) *Bio-Informatics Applications*

In the area of bioinformatics research, one might imagine a network of research data bases, belonging to specific governmental or private, non-for-profit research institutions, storing results or matching protein sequences and specific sub-sequences. The publishers in this case are the institutions' researchers publishing specific findings and subscribers are researchers inquiring for specific sequences' matching, as defined in their subscriptions.

Experimental Results

Typically, all proposed research solutions are accompanied by independent experimental results. So far the community has not produced widely acceptable general benchmarks, standard data sets, and workloads.

Data Sets

As mentioned, there are no generally-agreed upon real data sets to be used for testing research and development results. Some appropriate data sets are available, however, for some champion pub/sub applications, such as those based on RSS feeds. One example is data made available by the New York Stock Exchange (NYSE) (`http://www.nysedata.com/nysedata/default.aspx`).

Cross-references

- ▶ Channel-Based Publish/Subscribe
- ▶ Content-Based Publish/Subscribe
- ▶ Peer-to-Peer Content Distribution
- ▶ Peer to Peer Overlay Networks: Structure, Routing and Maintenance
- ▶ Peer-to-Peer System
- ▶ Publish/subscribe
- ▶ Publish/Subscribe over Streams
- ▶ Routing and Maintenance
- ▶ State-Based Publish/Subscribe
- ▶ Topic-Based Publish/Subscribe
- ▶ Type-Based Publish/Subscribe

Recommended Reading

1. Aberer K. P-Grid: a self-organizing access structure for P2P information systems. In Proc. Int. Conf. on Cooperative Inf. Syst., 2001.
2. Aekaterinidis I. and Triantafillou P. Internet scale string attribute publish/subscribe data networks. In Proc. Int. Conf. on Information and Knowledge Management, 2005.
3. Aekaterinidis I. and Triantafillou P. PastryStrings: a comprehensive content-based publish/subscribe DHT Network. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2006.
4. Banavar G., Chandra T., Mukherjee B., Nagarajarao J., Strom J., and Sturman D. An efficient multicast protocol for content-based publish-subscribe systems. In Proc. 19th Int. Conf. on Distributed Computing Systems, 1999.
5. Carzaniga A., Rosenblum D.S., and Wolf A.L. Design and evaluation of a wide-area event notification service. ACM Trans. Comput. Syst., 2001.
6. Castro M., Druschel P., Kermarrec A., and Rowstron A. Scribe: A large-scale and decentralized application-level multicast infrastructure. J. Select. Areas Commun., 2002.
7. Cugola G., Nitto E.D., and Fuggetta A. The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. In Proc. 23rd Int. Conf. on Software Eng., 2001.
8. Fabret F., Jacobsen A., Llirbat F., Pereira J., Ross K., and Shasha D. Filtering algorithms and implementation for very fast publish/subscribe. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001.

9. Gupta A., Sahin O.D., Agrawal D., and Abbadi A.E. Meghdoot: content-based publish subscribe over p2p networks. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004.
10. Lehman T., Laughry S., and Wyckoff P. Tspaces: The next wave. In Proc. 32nd Annual Hawaii Int. Conf. on System Sciences, 1999.
11. Pietzuch P.R. and Bacon J. Hermes: a distributed event-based middleware architecture. In Proc. 1st Int. Workshop Distributed Event-Based Systems, 2002.
12. Ratnasamy S., Francis P., Handley M., Karp R., and Shenker S. A scalable content addressable network. In Proc. ACM Int. Conf. on Data Communication, 2001.
13. Rowstron A. and Druschel P. Pastry: Scalable and distributed object location and routing for large-scale peer-to-peer systems. In Proc. IFIP/ACM Int. Conf. on Dist. Syst. Platforms, 2001.
14. Stoica I., Morris R., Karger D., Kaashoek F., and Balakrishnan H. Chord: A scalable peer-to-peer lookup service for internet applications. In Proc. ACM Int. Conf. on Data Communication, 2001.
15. Terpstra W.W., Behnel S., Fiege L., Zeidler A., and Buchmann A.P. A peer-to-peer approach to content-based publish/subscribe. In Proc. 2nd Int. Workshop Distributed Event-Based Systems, 2003.
16. TIBCO TIB/Rendezvous. Tech. rep., White paper, Palo Alto, CA, http://www.tibco.com, 1999.
17. Triantafillou P. and Aekaterinidis I. Publish-subscribe over structured P2P networks. In Proc. 3rd Int. Workshop Distributed Event-Based Systems, 2004.

Peer-to-Peer Storage

Anwitaman Datta
Nanyang Technological University, Singapore, Singapore

Synonyms

Distributed storage systems; Cooperative storage systems; Wide-area storage systems

Definition

Peer-to-peer (P2P) storage is a paradigm to leverage the combined storage capacity of a network of storage devices (peers) contributed typically by autonomous end-users as a common pool of storage space to store and share content, and is designed to provide persistence and availability of the stored content despite unreliability of the individual autonomous peers in a decentralized environment.

Historical Background

For diverse reasons including fault-tolerance, load-balance or response time, or geographic distribution of end users, distributed data stores have been around for a long while. This includes distributed databases, distributed file systems and *Usenet* servers among others. *Usenet* servers communicated among each other in a peer-to-peer manner, and replicated content.

While some redundancy is necessary for fault tolerance, replicating all content at all peers is a very special case of a peer-to-peer storage system, and is very inefficient. In general, an object is replicated at fewer locations. This leads to the problem of locating the object in the network. Plaxton et al's work on accessing nearby copies in a distributed environment [9] using key based routing is one of the seminal works, which subsequently influenced the design of many peer-to-peer storage systems. Advances in *structured overlay* networks address the problem of object location in a decentralized manner, that is "If an object is in the network, how to find it?" This is important in realizing an efficient decentralized peer-to-peer storage system.

Systems like *OceanStore* [7] aimed at archival storage, *Freenet* [3] for anonymous file sharing and distributed hash table based *DHash* storage layer of *CFS* [5] for a cooperative file system all store data at peers based on key associations determined by the structured overlay. These systems thus have the functionalities of object location and storage coupled together. However, note that locating and storing objects in a peer-to-peer system are in principle independent of each other. One can imagine the routing as a distributed index structure, which can store a pointer to the actual storage location, instead of the object itself. In the file sharing network *Napster*, the storage was peer-to-peer, however the indexing was in fact fully centralized, and very well illustrates the orthogonality of object location and storage issues. So in the rest of this entry the focus will be only on storage.

A critical requirement in a peer-to-peer storage system is to ensure that once a user (application) stores an object, this object should be available and persist in the network, notwithstanding the unreliability of individual peers and membership dynamics (churn) in the peer-to-peer network. This throws open a host of interesting design issues. For resilience, redundancy is essential. Redundancy can be achieved either by replication or using coding techniques, using schemes similar to the *RAID* technique [8]. While coding is in principle storage space efficient for achieving a certain level of resilience, it leads to various kinds of overheads, including computational overhead, and in the context of peer-to-peer systems, communication overhead and even

storage overhead to keep track of encoded object fragments, thus making coding mechanisms worthwhile only for relatively larger or rarely accessed objects and applications like archival storage.

Redundancy is lost unless replenished because of departure of peers from the system. Trade-off considerations of redundancy maintenance effort and resilience led to the design of different maintenance strategies [2,6,10].

Note that file sharing systems – early ones like *Napster* and *Gnutella* as well as more recent ones like *Kazaa* and *BitTorrent* – also store and provide access to stored objects. However they are not reliable storage systems. In file sharing networks, users store locally only files they are interested in, and allow others to download the same. However there is no explicit intention to provide a highly available and persistent storage, and hence there are no mechanisms for redundancy management. The design of these systems is often focused on improving the efficiency of search and data transfer, while the content available in the network is considered ephemeral. Nevertheless, popular content may get so widely replicated that it is coincidentally (but not by design) always available.

Foundations

Distributed storage systems have traditionally been managed in a centralized manner, for instance in distributed databases and distributed file systems. The advent of decentralized file sharing networks demonstrated the potential as well as feasibility of decentralized peer-to-peer storage systems composed of autonomous peers. Peer-to-peer storage systems use the combined capacity of the peers to provide storage functionality to end users.

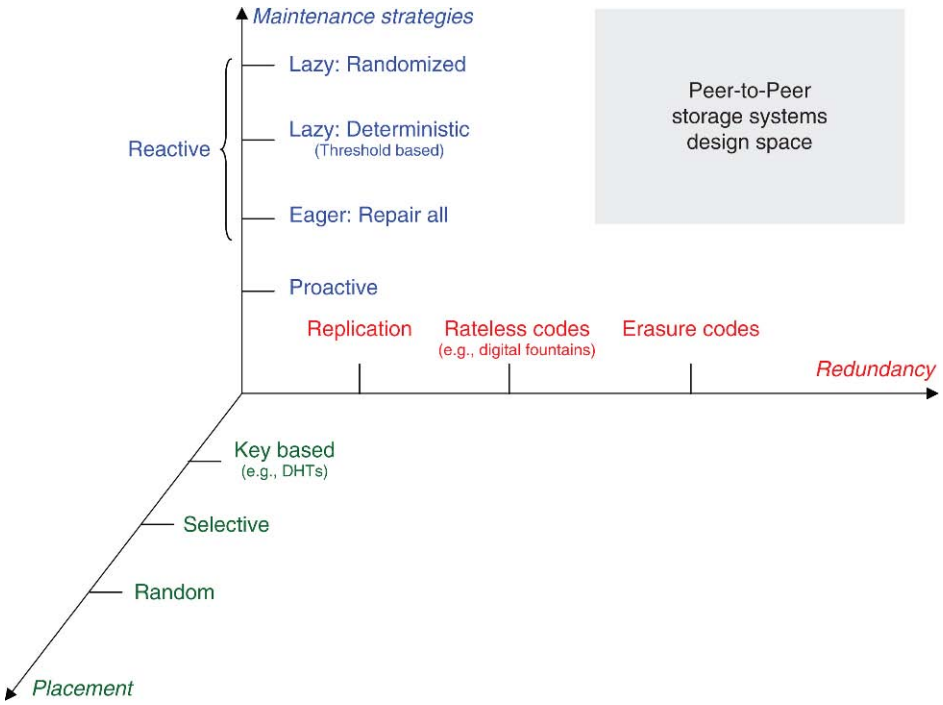
There are several reasons to have such a distributed storage. Multiple users may share and access some stored objects – data, files, etc. Individual users may not have the capacity to store all the objects they wish to access. Furthermore, by storing the same objects at other peers (and reciprocating by storing other users' objects), all peers benefit from an automatic back up service. Against these advantages, the main drawbacks include the unreliability of individual peers, who may leave and re-join autonomously, or even leave permanently, as well as transient communication failures, and delay in accessing objects stored only at a remote peer. Thus the main functionality of peer-to-peer storage systems is to make the distribution of stored objects transparent to the end users, even while users benefit from this distribution.

File sharing networks can be viewed as a special case of a peer-to-peer storage system. However they are not designed to guarantee availability or persistence of stored content, which are essential for a storage system. Resilience and performance in terms of access cost and latency pose some of the crucial challenges in realizing peer-to-peer storage systems.

Resilience of storage systems is measured in terms of two metrics - availability and durability (persistence). Availability of an object within a period of time is essentially the time averaged probability that it is accessible at any random time within that period of time. Durability of an object is the probability that it persists in the system indefinitely (or long enough, depending on the application requirements) under an assumption of a worst case failure scenario.

Consequently, crucial to peer-to-peer storage systems research are some of the following questions: What kind of redundancy is most efficient from various perspectives including storage overhead as well as access and maintenance costs and latency, and implementation complexity? What minimal redundancy is necessary to meet a desired level of resilience? Which maintenance strategy to apply to maintain the necessary redundancy? Which peers to store the redundant blocks in, based possibly on issues like reliability of individual peers, locality and load? The following delves into these. Figure 1 summarizes some of the important design factors.

In a decentralized setting, it is not only essential to ensure that stored objects stay available and persist over time despite changes in the network membership – churn – caused by peers leaving and (re-)joining the network, but also it is important to locate the stored objects efficiently. Some of the early storage systems like *OceanStore* [7] use one of the precursors [9] of contemporary structured overlays to locate objects. The basic idea is to assign to each peer an unique identifier, and also to each object an unique identifier from a same key-space, for instance using a hashing function. Objects are stored at peers which have identifiers closest to the object's identifier. When looking for an object, or the peer(s) at which an object is to be stored, the network of peers is typically searched in a greedy manner, by approaching peers with closer identifiers to the object's key, that is, using *key based routing*. Similar ideas are also used in *Freenet* [3], which aims at



Peer-to-Peer Storage. Figure 1. Some of the important design decisions that need to be taken into consideration when designing and deploying a peer-to-peer storage system.

providing anonymity to its users. Other subsequent storage systems have also followed this approach [5].

This traditional dual use of structured overlays for both routing (indexing) and storage tends to blur the difference between the two, and many people consider the structured overlay (e.g., distributed hash tables) to be the storage system also. In order to identify a full spectrum of design space of peer-to-peer storage systems it is crucial to understand that storage and indexing are actually two different but necessary ingredients for them. Combining these two sometimes simplifies the system architecture and implementation. However, the structured overlay may be used to store only pointers to the actual objects, which are placed using any independent criterion. One may also employ any other kind of directory (for example, *Napster* has a centralized directory) or indexing mechanism instead.

Resilience from Redundancy

Redundancy is essential to achieve resilience. Storage systems realize redundancy by using either replication or error correcting codes (e.g., erasure codes) as discussed next.

Replication Replication, i.e., mirroring, stores the same object on multiple nodes. It is normally efficient when the size of the object is small, or the object is frequently accessed. Note that a large object may still be split in multiple fragments, and each of these fragments will then be replicated.

Erasure codes Erasure codes are a class of error-correcting codes, which can transform a M-fragment object into N ($N > M$) encoded fragments, such that the original object can be reconstructed from any M *out of the* N encoded fragments. This typically leads to a storage overhead slightly more than N/M. The rate of erasure codes r is defined as the fraction of fragments required for decoding, i.e., $r = \frac{M}{N}$. Based on the coding rate, erasure codes can be categorized into two types, fixed-rate codes in which N is limited and r has a fixed value, and rateless codes (for example *Digital Fountains* [4]) in which N is potentially unlimited and r approaches zero (therefore called rateless).

Although replication is sometimes regarded as a special case of erasure codes, a subtle difference exists between them, which has its implications on the maintenance of redundancy. For replication, every reintegrated replica can enhance the object availability. In

contrast, for a fixed-rate erasure code, if the reintegrated fragment is identical to one of the existing fragments, it does not improve the availability of the whole object. Even with M fragments, one may still not be able to reconstruct the original object, unless the fragments are distinct. The above problem does not occur for rateless codes, in which unique fragments are generated, and thus every reintegrated fragment is useful. In this aspect, replication is more similar to rateless codes, instead of fixed-rate codes.

When accessing an object, if it had been stored in coded fragments, enough fragments need to be accessed first, and then the object needs to be decoded, causing computational and possibly communication overheads. The benefit of coding is that for the same targeted resilience, using coding techniques drastically reduces the storage overhead, alternatively said, for same storage overhead, coding provides a much higher resilience. However, depending on the implementation details, using coding techniques may also lead to additional storage overheads to keep track of the encoded fragments. For frequently accessed objects or small objects, the overheads associated with coding may thus outweigh the benefits.

Because of these considerations, a rule of thumb is small or frequently accessed objects are typically replicated, while large or rarely accessed objects erasure coded. Another option is to use a hybrid approach [11]. Erasure coded fragments are used to achieve high persistence at a low storage overhead. Replication is used for performance.

Maintaining Redundancy

Individual peers may be occasionally unavailable because users go offline, machines fail, or the network gets disconnected. Typically, if there is sufficient redundancy, such *temporary churn* has marginal effect on the availability of a stored object. Furthermore temporary churn does not directly affect long term persistence, since the peers join back, bringing back in the network whatever is stored in them. However, over a period of time, some participating peers may leave the system permanently, in turn leading to permanent loss of redundancy. *Permanent churn* thus makes the system more vulnerable to temporary churn, and unless mitigated, leads to permanent loss of stored objects, thus adversely affecting persistence/durability. So while redundancy provides resilience against temporary churn, maintenance strategies are necessary to restore the lost redundancy and make the system resilient against permanent churn.

Henceforth both coded fragments or replicas will be referred as fragments. The simplest maintenance mechanism is to probe periodically all the peers which are supposed to store redundant fragments of an object, and whenever a probe fails, reactively replenish the redundancy be reintegrating a new suitable redundant fragment. Example storage systems employing this strategy include CFS [5]. This strategy is referred to as an *eager maintenance* strategy. Such an eager reactive (The simultaneous use of the adjectives eager and reactive is admittedly oxymoronic, and is a vestige of the historical development and nomenclature of various maintenance mechanisms.) maintenance mechanism means the system always operates in a state where redundancy level remains constant apart from temporary reduction between repair periods. It has been empirically observed [2] that eager (reactive) maintenance strategy is bandwidth expensive.

A periodic probing and reintegration of all unavailable fragments ignores the fact that many of the fragments are only temporarily unavailable, and will be restored back in the system automatically once the corresponding peers join back. Consequently, a lot of overhead in regenerating and transferring new redundant fragments is avoidable if only the system can wait long enough for some of the absent peers to rejoin. This observation is key to the design of the first lazy repair (reactive) strategy, used in the *TotalRecall* system [2]. All nodes are probed periodically, and repairs are initiated only when less than a certain threshold $T_a > M$ of nodes (and corresponding data) are available. Thus to say, when an object has no more than T_a fragments available in the system, then a repair process for the object is initiated so that at the end of the repair process all N fragments are again available. As long as an object has a redundancy more than the parameter T_a, there is no maintenance. This lazy maintenance mechanism saves overheads caused by transient failures. However if the redundancy reduces below this threshold, then it is considered imprudent to further delay repair operations. So once the threshold is breached for an object, *TotalRecall* regenerates all the corresponding unavailable fragments. Thus, this reactive lazy mechanism has a deterministic trigger for initiating repairs.

While saving on bandwidth in comparison to the above mentioned eager reactive maintenance strategy, the threshold based lazy repair strategy suffers from several undesirable effects. First of all, by waiting for

the redundancy to fall below a threshold, the system is allowed to degenerate and become more susceptible as compared to other systems which have the same maximum redundancy N. Secondly, while most of the time this approach does not use the bandwidth even if it were available, once the threshold is breached, this approach tries to replenish all the missing fragments at once, thus causing bandwidth spikes. Finally, between the maintenance spikes, as redundancy falls, the available fragments are accessed more frequently, causing access load imbalance, particularly overloading the available peers. Two different works try to address these shortcomings.

A randomized variant of the lazy repair strategy [6] is to probe only a fraction of the stored fragments randomly (uniformly), until a minimal $T_b \geq M$ number of live fragments are detected. Thus a random number $T_b + X$ of probes (determined according to a probability distribution which in turn depends on the actual number of live fragments) will be required to locate T_b live fragments. Then X fragments which were detected to be unavailable are replaced by the system. The beauty of this randomized approach is that the expected value of X adapts with the number of live fragments. If there are fewer live fragments, then X will typically be large, and vice versa. Normally X can be typically much smaller than the total number of unavailable fragments at that instant. As a consequence, the repair process is continuous – thus available bandwidth is used more judiciously, avoiding spikes. The randomized repair process is naturally more aggressive when more redundant fragments are missing, while if very few fragments are missing then there are very few repairs.

While the randomized lazy repair strategy [6] tries to strike a balance between the periodic repair and the threshold based deterministic lazy repair strategies, and as a consequence the bandwidth usage is continuous and smoother, another independent approach [10] makes it an explicit goal to not exceed a bandwidth budget per unit time. Subject to the bandwidth budget per unit time it proactively creates new replicas. In contrast to the previously mentioned approaches all of which react to lost redundancy, the proactive approach does not aim to reduce overall bandwidth usage, but instead tries to ensure that by not exceeding the per unit time maintenance bandwidth budget, a better bandwidth provisioning can be achieved, and thus the maintenance operation does not interfere with applications. Another consequence of this approach is that typically enough redundancy is available to ensure equitable load distribution. However, a maximum redundancy threshold needs to be defined in using such a proactive replication approach to make a judicious use of the storage capacity.

Placement Strategies

A final system design issue is determining the placement strategy to be used to store the objects. One of the most widely used approach is to determine the placement of objects based on keys. In this scheme, all the peers as well as objects are assigned unique keys (e.g., by hashing), and then objects are stored at peers with closest or similar keys. The peers themselves communicate among each other by forming a structured overlay network, and messages are routed based on key similarity (key based routing). Such an object placement strategy can be seen in systems like *CFS* [5], *Oceanstore* [7], *Freenet* [3] and *Tempo* [10] to name a few. This approach essentially combines the storage of the object with the search mechanism.

Alternatively, object placement may be decoupled from the search mechanism, thus giving the systems designer, or even the applications using the storage system much more flexibility in choosing the storage location. This choice may be random (for example, in *TotalRecall* [2]), or based on reliability prediction derived from the history of peers' availability, or based on proximity (locality) from the end users accessing the object, load at peers, or other considerations like storing the object within a specific domain. Furthermore, this choice may be made by the peer-to-peer storage system designer or its administrator, or even independently by the applications and end users using the storage system.

Against this flexibility, the main disadvantage of decoupling storage from search is that one then needs some kind of directory service (potentially realized with a structured overlay) to perform the search functionality. The search provides pointer(s) to the stored object fragments. This creates additional storage and maintenance (of the pointers) overheads. Thus, in contrast to the key based storage approach, there is an additional level of indirection, and the storage system needs to explicitly keep track of any changes, including network level changes like change of peers' network address, unlike the structured overlay (key based storage) approach, where the structured overlay maintenance mechanism takes care of such changes and simplify storage systems design.

Analysis Techniques

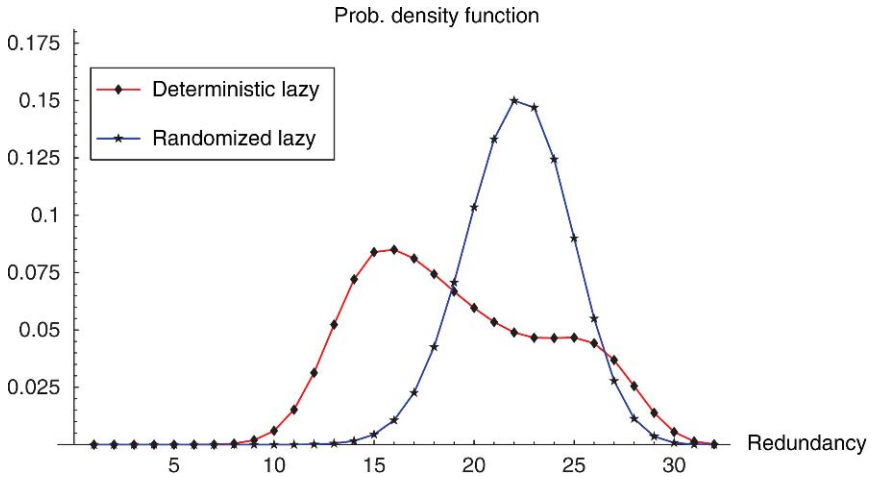
There are several approaches to analyze and study the behavior of peer-to-peer storage systems to better understand and refine its algorithms and design, make better parameter choices and validate its implementation.

Static resilience: Given a certain amount of redundancy, if a certain fraction of the peers are not accessible, either because they left the network, or the machines temporarily crashed, or because of communication problems, one can determine the probability that any specific object will become inaccessible or permanently lost. For example, for pure replication, if there are ρ replicas, and each peer storing a replica is available only p_{on} fraction of the time (randomly and independently from the other peers replicating the object), then the probability of the object becoming unavailable is $(1 - p_{on})^{\rho}$. Such a static resilience analysis gives a system designer a reference for determining an adequate redundancy to tolerate a certain degree of membership dynamics.

Time-evolution: Static resilience does not take into account the combined effect of membership dynamics, which leads to loss of redundancy over time, and repairing strategy, which regenerates redundancy, thus improving the health of the storage system. Given a particular level of network dynamics, the choice of maintenance strategy affects the resilience of the storage system, as well as the overheads incurred. A pragmatic system design thus needs to take into account the combined effect of individual peers' unreliability as well as the specifics of the deployed redundancy maintenance strategy.

Of particular interest is the actual probability distribution of object redundancy under the combined effect of churn and maintenance. This is in contrast to the maximum or average redundancy of objects in the system, based on which static resilience is typically estimated. Objects with smaller actual redundancy at a time instant are more vulnerable to become temporarily unavailable or even permanently lost. Also access to these corresponding objects causes higher load at the peers storing the object fragments or replicas. Finally, more effort and bandwidth is required to restore redundancy for the objects with fewer fragments, thus the maintenance operation witnesses spikes in bandwidth usage. The probability distribution provides a more fine-grained state of the storage system's health, by showing what fraction of all the stored objects are expected to have what level of actual redundancy. Such information can be obtained by studying the time evolution of storage systems [6,12].

For example, Fig. 2 shows the probability density function of the actual redundancy of objects when the



Peer-to-Peer Storage. Figure 2. A snapshot of the probability distribution of the available number of object fragments in the storage system when using (i) Deterministic lazy (reactive) maintenance, (ii) Randomized lazy (reactive) maintenance. An any *8 out-of 32* fragments (rate 0.25) erasure code was used to store the objects. Churn was simulated synthetically, such that online peers could go offline with a probability of 0.2 and offline peers could come back online with a probability of 0.1, such that on an average one third of the peers were online, i.e., $p_{on} = 1/3$. The thresholds T_a and T_b for the two variants of the lazy maintenance strategies were chosen such that the aggregate bandwidth usage for maintenance was the same in both experiments. This figure has been obtained from [6].

deterministic or the randomized lazy maintenance algorithms are used for an otherwise identical specific scenario, that is, same redundancy for objects, same churn level, and algorithm parameters chosen such that total bandwidth usage in both cases are comparable. There is a greater area under the curve corresponding to low redundancy if the deterministic lazy maintenance mechanism is used, in comparison to the case when the randomized variation is used.

Key Applications

Peer-to-peer storage systems have been used to realize traditional applications like file systems [2,5], backup [1] and archival storage [7]. A peer-to-peer backup system has several advantages in comparison to traditional offline backup systems based on secondary storage. It is easier to verify that the backup has indeed worked correctly (in comparison to physical medium like magnetic tapes), and backup as well as maintenance of the backed up data and its restoration whenever necessary can all be completely automated. Furthermore, peer-to-peer storage is not vulnerable to geographically localized catastrophes. This gives large corporations with geographically dispersed offices strong economic incentives to use their employees' desktop computers as a private corporate peer-to-peer storage infrastructure, instead of managing a separate secondary storage based backup. Similarly, individuals can back-up their data by relying on the resource pooled in a public peer-to-peer storage network.

Peer-to-peer storage systems also find use in web proxies for caching, content distribution and file sharing [3] applications. Peer data management systems too rely on an underlying reliable storage service.

Cross-references

- ▶ Peer Data Management System
- ▶ Peer-to-Peer Content Distribution
- ▶ Peer to Peer Overlay Networks: Structure, Routing and Maintenance
- ▶ Updates and Transactions in Peer-to-Peer Systems

Recommended Reading

1. http://www.cleversafe.org/dispersed-storage
2. Bhagwan R., Tati K., Cheng Y., Savage S., and Voelker G.M. TotalRecall: systems support for automated availability management. In Proc. 1st USENIX Symp. on Networked Systems Design & Implementation, 2004.
3. Clarke I., Miller S.G., Sandberg O., and Wiley B. Protecting free expression online using Freenet. IEEE Internet Computing, 6(1):40–49, 2002.
4. Codes R. and Shokrollahi A. IEEE Trans. Inform. Theory, 2006.
5. Dabek F., Kaashoek F., Karger D., Morris R., and Stoica I. Wide-area cooperative storage with CFS. In Proc. 18th ACM Symp. on Operating System Principles, 2001.
6. Datta A. and Aberer K. Internet-scale storage systems under churn – a study of the steady state using Markov models. In Proc. Sixth IEEE Int. Conf. on Peer-to-Peer Computing, 2006.
7. Kubiatowicz J., Bindel D., Chen Y., Czerwinski S., Eaton P., Geels D., Gummadi R., Rhea S., Weatherspoon H., Weimer W., Wells C, and Zhao B. OceanStore: an architecture for global-scale persistent storage. In Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operating Systems, 2000.
8. Patterson D., Gibson G.A., and Katz R. A case for redundant arrays of inexpensive disks (RAID). In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1988.
9. Plaxton C.G., Rajaraman R., and Richa A.W. Accessing nearby copies of replicated objects in a distributed environment. In Proc. ACM Symp. on Parallel Algorithms and Architectures, 1997.
10. Sit E., Haeberlen A., Dabek F., Chun B.G., Weatherspoon H., Morris R., Frans Kaashoek M., and Kubiatowicz J. Proactive replication for data durability. In Proc. 5th Int. Workshop Peer-to-Peer Systems, 2006.
11. Williams C., Huibonhoa P., Holliday J., Hospodor A., and Schwarz T. Redundancy management for P2P storage. In Seventh IEEE Int. Symp. on Cluster Computing and the Grid, (CCGrid), 2007.
12. Wu D., Tian Y., Ng K.-W., and Datta A. Stochastic analysis of the interplay between object maintenance and churn. Elsevier Journal of Computer Communications, Special Issue on Foundations of Peer-to-Peer Computing, Elsevier, 2007.

Peer-to-Peer System

Wojciech Galuba, Sarunas Girdzijauskas
EPFL, Lausaune, Switzerland

Synonyms

Peer-to-peer network; peer-to-peer overlay

Definition

A peer-to-peer system is a computer network which enables peers to share the network resources, computational power and data storage, without relying on a central authority. Most commonly, peer-to-peer systems form *overlay networks* deployed in the Internet and are used for file sharing, realtime data streaming and computationally intensive tasks.

Key Points

In contrast to client-server systems, peer-to-peer systems consist of interconnected peers of similar capabilities and responsibilities, where the peers can act as both servers and clients. Most commonly, the architecture of the peer-to-peer systems is flat and all peers are assumed to be functionally equal. However, a number of peer-to-peer systems employ hierarchical architecture where some peers (superpeers) act as local servers for the subsets of regular peers. It is also widely accepted in peer-to-peer systems, where some services can be provided by a centralized component, in particular system bootstrapping, authentication etc.

Peer-to-peer systems are designed to be self-organizing and to scale well with the number of participating peers. Each peer contributes a different amount of resources and peer-to-peer systems employ various methods to distribute the load evenly across these resources. Good load balancing is crucial to the scalability of peer-to-peer systems. To account for frequent peer departures and arrivals and to maintain high availability peer-to-peer systems replicate the services and data across several peers for redundancy.

Peer-to-peer systems have a wide range of applications in such areas as the Internet infrastructure (e.g., DNS), file sharing (e.g., Kazaa [3]), communication and data streaming (e.g., Skype [4]), distributed computing (e.g., BOINC [1]), and can even make use of human presence at the participating peers (e.g., Galaxy Zoo project [2]). In addition, the decentralized nature of peer-to-peer systems allows the peer-to-peer applications to be highly resistant to censorship.

High robustness, scalability and lack of the single-point-of-failure make peer-to-peer systems a viable alternative to large client-server systems.

Cross-references

► Distributed Hash Table

Recommended Reading

1. Berkeley Open Infrastructure for Network Computing Homepage. http://boinc.berkeley.edu/.
2. Galaxy Zoo Homepage. http://galaxyzoo.org/.
3. Kazaa Homepage. http://www.kazaa.com/.
4. Skype Homepage. http://www.skype.com/.

Peer-to-Peer Web Search

Gerhard Weikum
Max-Planck Institute for Informatics, Saarbrücken, Germany

Definition

The peer-to-peer (P2P) computing paradigm is an intriguing alternative to centralized search engines for querying and ranking Web content. In a P2P network with many thousands or millions of computers, every peer can have locally compiled content such as recently visited or thematically gathered Web pages, and can employ its own, potentially personalized search engine on the locally indexed data. In addition, queries can be forwarded to judiciously chosen other peers for collaborative evaluation. Such P2P architectures enable keyword search and result ranking on the network-wide global content. Thus, all peers together form a P2P search engine. Conversely, such P2P architectures could be utilized for scalable, distributed implementations of Web indexing with appropriate partitioning across the nodes of a large server farm.

Historical Background

Peer-to-peer (P2P) systems aim to provide scalable and self-organizing ways of loosely coupling thousands or millions of computers in order to jointly achieve some global functionality [14]. In the last decade, very successful systems of this kind have been built. Most notably, these include *file-sharing networks* such as Gnutella or BitTorrent, and IP telephony like Skype and other collaborative messaging services. They organize peers in so-called *overlay networks* on top of the standard Internet infrastructure, and use various forms of *epidemic dissemination* or distributed data structures like *distributed hash tables (DHTs)* such as Chord or Pastry. The basic functionality that underlies many of these systems is the distributed and dynamic maintenance of a dictionary with efficient support for exact-match key lookups.

Web search requires much richer functionality than exact-match lookups: keyword queries combine multiple dimensions of a very-high-dimensional data space in an ad hoc manner so that standard multi-dimensional indexes are not applicable, and they require ranking of query results based on statistics about local and global

keyword frequencies. On the other hand, it seems very natural to build Web search in a P2P manner as the producers and owners of Web pages are widely distributed and autonomous. In fact, the standard approach of crawling the Web for centralized search engines can be seen as rather artificial, but has advantages regarding system management and commercial services such as query-related advertisements.

Approaches to P2P Web search are reminiscent of earlier work on *distributed information retrieval (IR)*, most notably, various kinds of *metasearch engines* where queries are routed to judiciously chosen search providers [15]. However, the P2P setting is much more challenging regarding the enormous scale of the underlying data sources, the dynamics of the system, and the autonomy of the individual peers.

Foundations

Peer-to-peer (P2P) Web search has been studied from two perspectives: *global computing (GC)* and *social computing (SC)*. The GC approaches consider peers that dedicate all their storage and computing resources to the P2P network. The network as a whole emulates a traditional search engine architecture by partitioning a global Web index and assigning partitions and query load to peers. The SC approaches, on the other hand, consider architectures where each peer corresponds to one user, and emphasizes the autonomy of peers, with every peer having full control over its local content and extent of sharing it with other peers. Both GC and SC can be embedded in a server-farm environment with a high-speed physical network or in a geographically distributed environment over the wide-area Internet.

P2P Global Computing for Scalable Search-Engine Functionality

For *indexing the Web*, commercial search engines use large data centers consisting of thousands of low-end computers, carefully designed parallelism and redundancy, and customized software with very low overhead [2]. The key technique to sustain a peak throughput of thousands of queries per second with sub-second response times is *data partitioning*. Index entries – postings that consist of a keyword id, a page id, and a score – are hashed on page id, and each resulting hash partition is assigned to one computer in a very large *cluster* (with hundreds or thousands of nodes). A query with one or more keywords is simply sent to all nodes for parallel evaluation. The load is perfectly balanced across the nodes of the cluster, so that the approach scales up extremely well. For failure resilience, high availability, and higher throughput, an entire cluster is often replicated sufficiently.

The above architecture is distributed, but it is not a P2P approach, for it assumes a fixed, reasonably time-invariant system configuration with a high-speed homogeneous network between nodes. If one tried to carry this design over to a setting with a dynamically varying number of peers that communicate over a high-latency wide-area network, the fine-grained parallelism would probably result in a poor cost/performance ratio. Thus, a geographically distributed architecture for Web indexing needs more sophisticated techniques and is still viewed as a research challenge, but promising work is on its way [1].

The current approaches differ in their ways of partitioning the data across peers, the overlay networks that are employed, their load balancing mechanisms, and their methods for caching and replication, which in turn influences query processing strategies. As for *data partitioning*, three common ways are: hash-partitioning on page ids (documents), hash-partitioning on keywords (terms), or thematic clustering based on page contents or query logs. *Load balancing* needs to counter skewed distributions of both index-list lengths and query popularities for different keywords; many techniques from the distributed computing literature are applicable. *Caching* can consider different granularities like index lists for individual keywords or entire query results, and needs to employ appropriate strategies for cache refreshing and replacement. Finally, efficient algorithms for *distributed top-k query processing* are called for.

P2P Social Computing with Autonomous Peers

In the SC line of work, an architecture is pursued where every peer has a powerful local search engine, with its own crawler, indexer, and query processor. Such a peer can compile its own content from thematically focused crawls and other sources, and make this content available in a P2P overlay network. Search requests issued by a peer can first be executed locally, on the peer's locally indexed content. When the recall of the local search result is unsatisfactory, the query can be forwarded to a small set of other peers that are expected to provide thematically relevant, high-quality and previously unseen results. Deciding on this set of target peers is the *query routing* problem in a P2P search network, also known as *collection selection*. Subsequently, the actual

search on the chosen target peers requires efficient algorithms for *top-k query processing*. Search results are then returned by different peers and need to be meaningfully merged, which entails specific problems for *result ranking*. Both query routing and result ranking can build on various forms of *distributed statistics*, computed in a decentralized way and aggregated and disseminated in a scalable P2P manner (using compact synopses such as Bloom filters or hash sketches and leveraging the DHT infrastructure).

Query Routing

A practically viable query routing strategy needs to consider the similarity of peers in terms of their thematic profiles, the overlap of their contents, the potential relevance of a target peer's collection for the given query, and also the costs of network communication and peer-specific processing loads.

Traditionally, the most important measure for assessing the benefit of a candidate target peer for a given query is the estimated relevance of the peer's overall content for the query. This standard IR measure can be estimated frequency statistics over the query's keywords (terms in IR jargon). In conventional, document-oriented IR, these would be term frequencies (tf) within a document and the so-called inverse document frequencies (idf), the reciprocal of the global number of documents that contain a given term. In P2P IR, the estimation is based on the overall content of a peer as a whole. Instead of tf, the *document frequency (df)* of a peer is considered, which is the total number of documents that contain the term and are in the peer's collection; and instead of idf, the *inverse collection frequency (icf)* is considered, which is the reciprocal of the total number of peers that contain (at least one document with) the term. These basic measures are combined into a relevance or query-specific quality score for each candidate peer. There are various models for the combined scores; among the most cited and best performing models are *CORI* [4], based on probabilistic IR, and statistical language models adapted to the setting of P2P collection selection [9]. The Decision-Theoretic Framework (DTF) [11] provides a unified model for incorporating quality measures of this kind as well as various kinds of cost measures.

Selecting peers solely by query relevance, like CORI routing, potentially wastes resources when executing a query on multiple peers with highly overlapping contents. To counter this problem, methods for estimating the overlap of two peers' contents have been developed. These estimates are then factored into an *overlap-aware query routing* [10] method by using a weighted combination of peer quality and overlap (or novelty) as ranking and decision criterion.

Query routing decisions are typically made at query run-time, when the query is issued at some peer. But the above methods involve directory lookups, statistical computations, and multi-hop messages; so it is desirable to precompute preferred routing targets and amortize this information over many queries. A technique for doing this is to encode a similarity-based precomputed binary relation among peers into a *Semantic Overlay Network (SON)* [5]. The routing strategy would then select target peers only or preferably from the SON neighbors of the query initiator.

Search Result Ranking

When a query returns results that have been obtained from different peers, the scores that the peers assign to them are usually not comparable. The reason is that different peers may use different statistics, for example, for estimating the idf value of a term which is crucial for weighting the importance of different query terms, or they may even use completely different IR models. This situation leads to the problem of *result merging*. It is addressed by re-normalizing scores from different peers to make results meaningfully comparable [15]. A variety of such methods exist in the literature, some using only the peer-specific scores and some aggregated measures about peers (e.g., the total number of documents per peer), some using sampling-based techniques, and some using approaches that first reconstruct the necessary global statistics (e.g., global document frequencies for each term) for optimal re-normalization of scores.

Web search ranking usually also considers the query-independent authority of pages as derived from link analysis, and a P2P network is a natural habitat for such "social ratings". *Link analysis* algorithms such as PageRank are centralized algorithms with very high memory demand. Executing them in a distributed manner would allow scaling up to even larger link graphs, by utilizing the aggregated memory of a P2P system. Various decentralized methods have been developed to this end, including a general solution to the spectral analysis of graphs and matrices [8], which underlies the PageRank computation.

Most of these methods assume that the underlying Web graph can be nicely partitioned among peers. In contrast, a P2P system with autonomous peers faces a situation where the Web pages and links that

are known to the individual peers are not necessarily disjoint. The JXP algorithm [12] computes global authority measures such as PageRank in a decentralized and scalable P2P manner, when the Web graph is spread across autonomous peers and peers' local graph fragments overlap arbitrarily, and peers are (a priori) unaware of other peers' fragments. The scores computed by JXP provably converge to the same values that would be obtained by a centralized PageRank computation on the full Web graph. These kinds of algorithms seem to be highly relevant also for analyzing authority and reputation measures in large-scale social networks.

Key Applications

The technology for P2P Web search has many potential applications, ranging from keyword search on the Web or in blogs, possibly in a personalized manner and exploiting social-network affinities among users, all the way to more "semantic" search capabilities in large enterprises, scholarly communities, federations of digital libraries, and distributed Internet archives. The latter may include searching XML, RDF, or multimedia data as well as providing temporal querying and other forms of enhanced search capabilities. With richer functionality, centralized systems face scalability bottlenecks, whereas decentralized approaches can leverage the fact that the underlying information is naturally distributed at a large scale.

Future Directions

Today, there is still no P2P Web search system that would scale anywhere near the sizes of the major commercial search engines. But as the Web continues to grow, search functionality is becoming richer (e.g., by personalization), and workloads are becoming much more demanding, the P2P paradigm is likely to gain more momentum for Web applications. Currently, there is a variety of research prototypes (e.g., [3,6,7,13]), including some that offer open-source software.

Cross-references

- ▶ Distributed Hash Table
- ▶ Link Analysis
- ▶ Peer-to-Peer Data Management
- ▶ Social Networks
- ▶ Top-k XML Query Processing

Recommended Reading

1. Baeza-Yates R.A., Castillo C., Junqueira F., Plachouras V., and Silvestri F. Challenges on distributed web retrieval. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 6–20.
2. Barroso L.A., Dean J., and Hölzle U. Web search for a planet: the Google cluster architecture. IEEE Micro, 23(2):22–28, 2003.
3. Bender M., Michel S., and Parreira J.X., and Crecelius T. P2P web search: make it light, make it fly. In Proc. 3rd Biennial Conf. on Innovative Data Systems Research, 2007, pp. 164–168.
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. Crespo A. and Garcia-Molina H. Semantic overlay networks for P2P systems. In Proc. 3rd Int. Workshop Agents and Peer-to-Peer Computing, 2004, pp. 1–13.
6. Cuenca-Acuna F.M., Peery C., Martin R.P., and Nguyen T.D. PlanetP: using gossiping to build content addressable peer-to-peer information sharing communities. In Proc. 12th IEEE Int. Symp. High Performance Dist. Comp., 2003, pp. 236–249.
7. Kalnis P., Ng W.S., Ooi B.C., and Tan K.-L. Answering similarity queries in peer-to-peer networks. Inf. Syst., 31 (1):57–72, 2006.
8. Kempe D. and McSherry F. A decentralized algorithm for spectral analysis. In Proc. 36th Annual ACM Symp. on Theory of Computing, 2004, pp. 561–568.
9. Lu J. and Callan J.P. Content-based retrieval in hybrid peer-to-peer networks. In Proc. Int. Conf. on Information and Knowledge Management, 2003, pp. 199–206.
10. Michel S., Bender M., Triantafillou P., and Weikum G. IQN routing: integrating quality and novelty in P2P querying and ranking. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 149–166.
11. Nottelmann H. and Fuhr N. Comparing different architectures for query routing in peer-to-peer networks. In Proc. 28th European Conf. on IR Research, 2006, pp. 253–264.
12. Parreira J.X., Donato D., Michel S., and Weikum G. Efficient and decentralized pageRank approximation in a peer-to-peer web search network. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 415–426.
13. Podnar I., Rajman M., Luu T., Klemm F., and Aberer K. Scalable peer-to-peer web retrieval with highly discriminative keys. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 1096–1105.
14. Steinmetz R. and Wehrle K. Peer-to-peer systems and applications. Springer, 2005.
15. Weiyi M., Yu C.T., and Liu K.-L. Building efficient and effective metasearch engines. ACM Comput. Surv., 34(1):48–89, 2002.

Performance Analysis of Transaction Processing Systems

Alexander Thomasian
Thomasian and Associates, Pleasantville, NY, USA

Synonyms

Queueing analysis; Probabilistic analysis; Cache performance; Storage systems; Concurrency control

Definition

The performance of *transaction (txn) processing (TP)* systems and more generally *database management systems (DBMSs)* is measured on operational systems, prototypes, and benchmarks. Probabilistic and queueing analyses have been used to gain insight into TP system performance, but also to develop capacity planning tools. The following is considered: (i) queueing analysis of processors and disks, (ii) *queueing network models (QNMs)* of computer systems, (iii) techniques to estimate the database buffers miss rate, (iv) factors affecting RAID performance, (v) *concurrency control (CC)* methods for high data contention TP systems and their analyses.

Historical Background

Early performance studies of TP were concerned with processor or *central processing unit (CPU)* scheduling. *Queueing network models – QNMs* were developed in the 1970s to estimate delays at active computer system resources, i.e., CPU and disks. The effect of passive resources, such as the memory size constraint, was later incorporated into capacity planning tools for TP systems [5]. The *Transaction Processing Council's (TPC's)* debit–credit benchmark in 1985 compares the performance of TP systems using their throughput at which a certain txn response time is reached (http://www.tpc.org). Buffer/cache management policies in processors, databases, and disk controllers have been studied since the 1970s. Coherence issues of CPU caches in multiprocessors and database buffers in shared disk systems gained importance in 1980s. Magnetic disks invented in 1950s have a significantly improved capacity and transfer rate, but not random access time. This led to the proposal and analysis of numerous disk arm scheduling policies. The 1988 *Redundant Array of Independent Disks (RAID)* classification provided renewed impetus to improve the performance and reliability of storage systems [1]. Shared-everything, -disk, and -nothing computer organizations have limitations for high-performance TP and DBMS applications, i.e., the first two pose cache coherence problems and shared-nothing systems are susceptible to processing time skew. (For example, there is a twofold increase with exponentially distributed processing times at four nodes, since $\sum_{i=1}^{4} 1/i = 2.08$.) CC has limited effect on TP performance in modern DBMSs, but this was not so in early relational DBMSs with coarse granularity locking, therefore many CC methods were proposed and evaluated [10].

Foundations

CPU Scheduling. Processing of txns at the CPU has been modeled by an M/G/1 queueing model, which consists of a *First-Come, First-Served (FCFS)* queue and a single server [4]. M stands for Poisson arrivals with rate λ, G stands for a general service time distribution with $\overline{x^i}$ as the ith moment. The server utilization is $\rho = \lambda\overline{x} < 1$, the mean waiting time at the queue is $W_{M/G/1} = \lambda\overline{x^2}/(2(1-\rho))$, and the mean response time $R = W_{M/G/1} + \overline{x}$ [4]. M/M/1 is a special case of M/G/1 with an exponential service time distribution with $\overline{x^2} = 2(\overline{x})^2$, so that the mean of the exponentially distributed response time is $R_1 = \overline{x}/(1-\rho)$. M/M/m has m servers, hence $\rho = \lambda\overline{x}/m$. For $m = 2$ there is $R_2 = \overline{x}/(1-\rho^2)$ (for $m > 2$ see [4]). For a single-server which is twice as fast as the previous servers: $R_3 = (\overline{x}/2)/(1-\rho)$. The same ρ is maintained in the three systems with R_i, $1 \le i \le 3$, by setting the arrival rate to 2λ, but utilizing two M/M/1 queues with uniform routing in the first case, so that the total processing capacity is 2μ in all three cases. There is $R_1 > R_2 > R_3$, where the first inequality reflects the resource sharing advantage of a shared queue, while the second inequality reflects the advantage associated with a single fast server when service times are not highly variable [4]. For K fork-join requests initiated at K M/M/1 queues the expected value of the maximum of K response times is: $R_k^{max} = H_K R$, where $H_K = \sum_{k=1}^{K} 1/k$ is the Harmonic sum. The components of a fork-join request are correlated, so that $R_2^{F/J} < R_k^{max}$, e.g., $R_2^{F/J} = (1.5 - \rho/8)R < R_2^{max}$. An approximate expression for $R_k^{F/J}$ for $K > 2$ has been used in the analysis of RAID5 disk arrays. With two txn classes the response time for the more urgent (class 1) txns can be improved by processing them at a higher priority than less urgent (class 2) txns. The arrival rates are λ_1 and λ_2 and the ith moments of service time are $\overline{x_1^i}$ and $\overline{x_2^i}$, respectively. With preemptive priorities and a negligible preemption overhead, class 1 txns are processed without being affected by class 2 txns. With non-preemptive priorities $W_1 = W_0/(1-\rho_1)$ and $W_2 = W_0/((1-\rho_1)(1-\rho))$, where $\rho_j = \lambda_j\overline{x}_j$, $j = 1,2$ and $W_0 = \frac{1}{2}\sum_{j=1}^{2}\lambda_j\overline{x_j^2}$ is the remaining processing time of txns at the CPU [4]. Note that W_1 is only affected by ρ_1, but not $\rho = \rho_1 + \rho_2$. Kleinrock's *conservation law* states that the improvement in waiting

time of one txn class is at the cost of the other, so that the weighted sum of waiting times remains constant: $\sum_{j=1}^{2} \rho_j W_j = W_0/(1-\rho)$. Preemptive (resp. non-preemptive) priorities are applicable to CPU (resp. disk) scheduling.

Queueing Network Models – QNMs

In *open QNMs* there are txn arrivals and departures, while in a *closed QNM* a completed txn is immediately replaced by a new txn, so that the number of txns remains fixed at N. Txns are processed at $K > 1$ nodes, where each node is a single or multiserver queueing system. The QNM of a computer system may be organized as the *Central Server Model (CSM)*, with the CPU the central server and the disks as peripheral servers. Txns arrive at the CPU (node 1), access one of the $K-1$ disks with probability p_k, $2 \le k \le K$, return to the CPU for additional processing, and leave the system after CPU processing with probability p_1, so that $\sum_{k=1}^{K} p_k = 1$. The transition probability matrix yields the mean number of visits to the nodes: i.e., $v_1 = 1/p_1$ and $v_k = p_k/p_1, 2 \le k \le K$. QNMs satisfying the BCMP theorem allow four types of nodes, most notably exponential service times at single- or multi-server queues with FCFS scheduling and nodes with an infinite number of servers [4,5]. The steady-state probability $Pr[n_1,n_2,...,n_K]$ in such QNMs can be expressed in product-form and their performance metrics computed efficiently. Approximation techniques or simulation can be applied otherwise.

The mean residence times at the nodes of an open product-form QNM can be obtained separately. With an external arrival rate Λ, the arrival rate to node k is $\lambda_k = v_k\Lambda$, its utilization $\rho_k = \lambda_k \overline{x}_k/m_k$, where $\overline{x}_k$ is its mean service time and m_k is the number of its servers. The service demand or the total txn processing time at a node k is given as $D_k = v_k\overline{x}_k$, $1 \le k \le K$, so that alternatively $\rho_k = \Lambda D_k$. The mean txn residence time at node k is $r_k = \overline{x}_k/(1-\rho_k^{m_k})$, $1 \le k \le K$ with $m_k \le 2$. The mean txn response time is $R = \sum_{j=1}^{K} v_k r_k$ and the mean number of txns at the system following Little's result is the product of the arrival rate and the mean response time: $\overline{N} = \Lambda R$ [4,5]. $\overline{N}$ can be used to estimate the degree of txn concurrency or *Multiprogramming Level (MPL)* and the memory size requirements for TP.

Closed QNMs can be analyzed via the convolution or the *mean value analysis – MVA* algorithm [4,5]. The MVA *arrival theorem* states that in a closed QNM with n txns the mean queue-length at node k at an arrival instant, is that of a closed system with $n-1$ txns, i.e., $A_k(n) = Q_k(n-1)$ [5]. The mean queue-length includes the request at the server, whose mean residual service time is the same as its service time due to the memoryless property of the exponential distribution: $\overline{x'}_k = \overline{x^2}_k/(2\overline{x}_k) = 2(\overline{x}_k)^2/(2\overline{x}_k) = \overline{x}_k$ [4]. The mean residence time at node k for v_k visits is $R_k(n) = v_k r_k(n) = v_k\overline{x}_k[1+Q_k(n-1)] = D_k[1+Q_k(n-1)]$.

MVA Algorithm for Closed QNM with Single Servers.
Input parameters: N (MPL), K (number of nodes), D_k, $1 \le k \le K$ (mean service demands).
for $k = 1$ to K do $Q_k(0) = 0$ (initialize queuelengths at $n = 0$)
for $n = 1$ to N do {(vary the number of txns)
for $k = 1$ to K do {(vary the node index)
if delay server $R_k(n) = D_k$ (no queueing at "infinite servers")
if single server $R_k(n) = D_k[1 + Q_k(n-1)]$}
$R(N) = \sum_{k=1}^{K} R_k(n)$ (txn residence time)
$X(n) = n/R(n)$ (txn throughput using Little's result)
for $k = 1$ to K do $Q_k(n) = X(n)R_k(n)$ (mean queue-length at the nodes, Little's result) }

The convolution algorithm applied to single-server queues computes $G(0:N)$, after the initialization $G(0) = 1$ and $G(n) = 0$, $n \ne 0$, as: $G(n) = G(n) + D_k G(n-1)$, $1 \le n \le N$, $1 \le k \le K$, so that $X(N) = G(N-1)/G(N)$. Compared to MVA the order of the iterations on n and k are reversed. Given that $\sum_{k=1}^{K} D_k$ =constant, the throughput of a closed QNM is maximized when the service demands at all the nodes are equal. Due to symmetry $Q_k(N) = (N-1)/K$, so that $R(N) = KD(1+(N-1)/K) = (N+K-1)D$. Next, consider Poisson arrivals to a QNM with a maximum degree of concurrency M due to limited memory size. For $n < M$ arriving txns are activated immediately and otherwise enqueued at a FCFS memory queue. A hierarchical solution method is applicable in this case [5]. The lower level model yields the throughput characteristic of the closed QNM: $X(n)$, $1 \le n \le M$. Additional txns waiting in the memory queue do not contribute to throughput, so that $X(n) = X(M)$, $n > M$. A birth-death model is used at the higher level, with the state n denoting the number of txns at the system. The arrival rate at all states is Λ and the completion rate at state n is $X(n)$, $n \ge 1$. The probability that there are n txns in the system is: $P_n = \Lambda P_{n-1}/X(n)$, $n \ge 1$. The normalization condition $\sum_{n\ge 0} P_n = 1$ yields P_0 and

hence $P_n, \forall n > 0$. The mean number of txns at the computer system, mean number of txns at the memory queue, and the mean number of activated txns is: $\overline{N} = \sum_{n>0} nP_n$, $\overline{N}_B = \sum_{n>M}(n-M)P_n$, and $\overline{N}_A = \sum_{n>0} min(n, M)P_n$. The mean txn response time, mean waiting time at the memory queue, and the mean residence time at the computer system are, respectively: $R = \overline{N}/\Lambda$, $W_M = \overline{N}_B/\Lambda$, and $R_A = \overline{N}_A/\Lambda$. $\overline{N} = \overline{N}_B + \overline{N}_A$, so $R = W_M + R_A$. The throughput characteristic $X(n)$, $n \geq 1$ vs. n (without an MPL constraint) is a convex function, which reaches an asymptotic value $min\{m_k/D_k, \forall k\}$. The node with the smallest throughput is the bottleneck resource, since it determines the maximum system throughput Λ_{max}. The throughput may drop due to thrashing in an overloaded virtual memory system [5] or due to excessive lock contention with the standard locking method.

Buffer Miss Rate (BMR)

BMR is applicable to the various levels of the memory hierarchy. Misses at the database buffer and disk controller cache result in disk accesses, which significantly degrade the performance of TP systems. BMR is affected by the buffer size, the replacement policy, and the workload. Analyses of replacement policies, such as FIFO, LRU, and CLOCK, using Markov chain modeling are based on the *Independent Reference Model (IRM)*. (http://www.informatik.uni-trier.de/~ley/db/dbimpl/buffer.html.) Data streams can also be characterized by stack depth and the hierarchical reuse or fractal models. Trace-driven simulation is a straightforward technique for evaluating the performance of buffer management policies, which can be used for calibrating empirical formulas. A recent top-down analysis derives an equation for the number of page faults (n) vs. memory size (M) [9]. M_0 and M^* are the min and max M that are sufficient and necessary. n^* is the number of cold memory misses, i.e., misses starting with an empty buffer. The parameters vary with program behavior and replacement policy.

$$n = \frac{1}{2}[H + \sqrt{H^2 - 4}](n^* + n_0) - n_0,$$
$$H = 1 + \frac{M^* - M_0}{M - M_0}, M \leq M^*.$$

Performance Analysis of Storage Systems

RAID performance is improved by striping and caching. Striping eliminates disk access skew by partitioning large datasets into fixed size *stripe units (SUs)*, which are allocated in round-robin manner across all disks [1]. The RAID controller cache complementing the database buffer in main memory is partially *nonvolatile storage (NVS)* and is used for *fast writes*, which are as reliable as writing to disk. Disk writes can be deferred and processed at a lower priority than reads, since read response times affect txn response time. Dirty data blocks in NVS are possibly overwritten several times, before they are destaged or written to disk. This eliminates unnecessary disk accesses. Applying disk scheduling to batched destaging of multiple blocks leads to higher disk access efficiency. Both effects can be captured by trace analysis [15].

RAID1 or disk mirroring, which predates the RAID classification, replicates data on two disks to achieve fault-tolerance. A data block can be read from either disk, but both copies should be eventually updated. Disk bandwidth can be saved by writing modified blocks to NVS first. The doubling of data access bandwidth is beneficial in view of increasing disk capacities, but an even further improvement in bandwidth can be attained by judicious routing of requests. When one of mirrored disks fails the read load on the surviving disk is doubled, but a smaller load increase can be attained with *interleaved declustering*, which replicates the contents of one disk on the other $n-1$ disks of the cluster, so that the read load increase is $n/(n-1)$ [13].

Erasure coding in RAID5 and RAID6 uses one and two check disks to protect against single and double disk failures, respectively [1]. RAID5 uses parity coding, so that the check SU is the *exclusive-OR (XOR)* of the remaining SUs in the same stripe or row. RAID6 uses Reed-Solomon or specialized parity codes to protect two disks out of N. The updating of a data block requires the corresponding check block(s) to be computed and updated. If the old copies of data and check blocks are not cached, four (resp. six) disk accesses are required for RAID5 (resp. RAID6), this is therefore referred to as the *small write penalty* [1]. Check SUs are placed in repeating left-to-right diagonals to balance the load for updating the check blocks. When one of the N disks fails, RAID5 continues its operation in degraded mode, by reconstructing missing blocks of the broken disk on demand as follows. The controller issues a fork-join request to the $N-1$ corresponding blocks on surviving disks and these blocks are XORed to reconstruct the missing block. The read load on surviving disks is doubled, but the load increase for

write requests is smaller. The *clustered RAID* paradigm reduces the disk load in degraded mode by using a parity group size $G < N$, so that the increase in read load is $\alpha = (G-1)/(N-1) < 1$ [1]. The RAID5 rebuild process reconstructs the contents of a failed disk track-by-track on a spare disk, by reading successive tracks from surviving disks, XORing all corresponding tracks to reconstruct a missing track, and writing it on the spare disk [15]. Distributed sparing distributes the spare areas among $N+1$ disks. so that the bandwidth of the spare disk is not wasted [15]. The *vacationing server method (VSM)* starts reading tracks from disks when they become idle, so the rebuild process does not affect the disk load. No new tracks are read after there is an external arrival, but the reading of the current track is completed. The increase in mean disk waiting time over M/G/1 is the mean residual time to read a track: $W_{VSM} = W_{M/G/1} + \overline{x'}_{track}$ [4]. RAID6 deals with the possibility of unsuccessful rebuild, when unreadable sectors or *latent sector failures – LSFs* are encountered during the rebuild process or to cope with rare two disk failures.

Mean disk response time with FCFS disk scheduling can be obtained using the M/G/1 queuing model [4]. Random disk accesses incur a seek, latency or rotational delay (half a rotation time for small blocks), and transfer time (negligible for small blocks). One has $x_{disk} = x_{seek} + x_{latency} + x_{xfer}$. The moments of disk access time can be obtained under various assumptions, e.g., all disk cylinders are accessed uniformly. A significant reduction in disk response time is possible via the *Shortest Access Time First (SATF)* policy [14]. Given n random disk requests, the disk access time drops as $n^{1/5}$, e.g., $n = 32$ halves the access time with respect to FCFS.

Performance Analysis of Concurrency Control Methods

Performance degradation due to CC is in the form of txn blocking and restarts. The *standard locking* method is based on the strict *two-phase locking (2PL)* method [3], i.e., locks are requested on demand and held to completion, at which point the txn commits, releasing all of its locks. Locks are also released when a txn is aborted to resolve deadlocks or other reasons [3]. Restart-oriented locking methods and *optimistic CC (OCC)* methods are also discussed.

Standard Locking

Consider an abstract model of standard locking [12]. There are M txns in a closed system. A txn of size k has $k+1$ steps with mean duration s, so that the mean txn residence time with no lock contention is: $r_k = (k+1)s$. A lock is requested at the end of the first k steps and all locks are released at the end of the $k+1$st step, i.e., strict 2PL, so that lock requests are uniformly distributed over the lifetime of a txn. The mean number of locks held by a txn is the ratio of the time–space of the number of locks held by a txn and r_k, which is an approximation to R_k at lower levels of lock contention, which yields: $\overline{L}_k \approx k/2$. Assuming that all lock requests are exclusive and are uniformly distributed over the D objects in the database, the probability of lock conflict is: $P_c(k) \approx (M-1)\overline{L}_k/D$. When the fraction of shared lock requests is f_S, the analysis can proceed as if all the lock requests are exclusive with an effective database size $D_{eff} = D/(1-f_S^2)$ [8]. The probability that a txn encounters a lock conflict for $P_c(k) \ll 1$ can be used to obtain the probability of a two-way deadlock [3]:

$$P_w(k) = 1 - (1-P_c(k))^k \approx kP_c(k) \approx \frac{(M-1)k^2}{2D}.$$

$$P_{D(2)}(k) = \frac{1}{M-1} Pr[T_1 \to T_2] Pr[T_2 \to T_1] = \frac{(P_w(k))^2}{M-1} \approx \frac{(M-1)k^4}{4D^2}.$$

A two-way deadlock can only occur if txn T_A requests a lock held by txn T_B, which is blocked because of its previous lock conflict with T_A. Multiway deadlocks are much less common and are ignored [3]. We obtain the mean blocking delay of a txn with respect to an active txn holding the requested lock ($W_k^{(1)}$) by noting that the probability of lock conflict increases with the number of locks held by the txn (j) and that there are $(k-j)$ remaining steps each with mean duration $(s+u)$ [10,11]:

$$W_k^{(1)} \approx \sum_{j=1}^{k} \frac{2j}{k(k+1)} [(k-j)(s+u)] + s' = \frac{k-1}{3}[r+u] + s'.$$

$u = P_c(k)W_k$ is the mean blocking time per step, where W_k is the mean waiting time per lock conflict, and s' is the remaining processing time of a step. Since deadlocks are rare, the mean txn response time only takes into account txn blocking time due to lock conflicts: $R_k \approx (k+1)s + kP_c(k)W_k$. The fraction of txn blocking time per lock conflict with an active txn is $A = W_k^{(1)}/R_k \approx 1/3$. A more accurate expression for

two-way deadlocks as verified by simulation is: $P'_{D(2)}(k) \approx AP_{D(2)}(k) = P_{D(2)}(k)/3$ [16]. Next consider K txn classes in a closed system, so that a completed txn is replaced by a class k txn with frequency $f_k, \sum_{k=1}^{K} f_k = 1$. The mean response time over all txn classes is: $R = (K_1 + 1)s + K_1 P_c W$, where $K_i = \sum_{k \geq 0} k^i f_i$ is the ith moment of txn size. The mean number of txns in class k is $\overline{M}_k = MR_k f_k / R$ (the summation $\sum_{k=1}^{K}$ yields M on both sides). Given that $\overline{M}_k = M(k+1)f_k/(K_1+1) \approx Mkf_k/K_1$, the mean number of locks held per txn is then:

$$\overline{L} = \frac{1}{M}\sum_{k=1}^{K} \overline{L}_k \overline{M}_k \approx \frac{1}{M}\sum_{k=1}^{K} \frac{k}{2}\overline{M}_k \approx \frac{1}{2K_1}\sum_{k=1}^{K} k^2 f_k = \frac{K_2}{2K_1}.$$

Similarly to fixed size txns $P_c \approx (M-1)\overline{L}/D$ and the expression for $P_{D(2)}$ is given in [16,12]. Both P_c and $P_{D(2)}$ are affected by the variability of txn sizes, e.g., for a truncated geometric distribution there is a two-fold and tenfold increase with respect to fixed txn sizes, respectively [16]. W_1 for variable txn sizes can be obtained as the expected value of $W_1(k)$ over all txn sizes. The normalized value for W_1 is [10,11]:

$$A = \frac{W_1}{R} \approx \frac{K_3 - K_1}{3K_1(K_2 + K_1)}.$$

Denote the fraction of blocked txns with β and ignore the difference in the number of locks held by active and blocked txns. The probability that a txn has a lock conflict with an active txn at level $i = 0$ (resp. has a lock conflict with a txn blocked at level $i = 1$) is $1 - \beta$ (resp. β) and the mean blocking time is W_1 (resp. $W_2 = 1.5W_1$). The expression for W_2 relies on the fact that at the random instant when the lock conflict occurs the active txn in the waits-for-graph is halfway to its completion. Generally, the probability that a txn is blocked at level i is: $P_b(i) \approx \beta^{i-1}, i > 1$ and $P_b(1) = 1 - \beta - \beta^2 - \beta^3 \ldots$. The mean waiting time at level $i > 1$ is $W_i \approx (i - 0.5)W_1$, which is motivated by the expression for W_2. The mean blocking time is a weighted sum of delays incurred by txns blocked at different levels:

$$W = \sum_{i \geq 1} P_b(i) W_i = W_1 [1 - \sum_{i \geq 1} \beta^i + \sum_{i > 1} (i - 0.5)\beta^{i-1}].$$

Let $\overline{M}_A$ and $\overline{M}_B$ denote the mean number of active and blocked txns, so that $\overline{M}_A + \overline{M}_B = M$. The fraction of blocked txns is $\beta = \overline{M}_B / M$. Dividing $\overline{M}_B$ and M by the system throughput yields $\beta = K_1 P_c W / R$. Multiplying both sides of the above equation by $K_1 P_c / R$ yields β on the left hand side and $\alpha = K_1 P_c W_1 / R = AK_1 P_c$ on the right hand side. Note that α is the product of the mean number of lock conflicts per txn and the normalized waiting time with respect to active txns. It follows: $\beta = \alpha(1 + 0.5\beta + 1.5\beta^2 + 2.5\beta^3 + \ldots)$. A closed-form expression for β can be obtained by noting that $\beta < 1$ and assuming that the series is infinite:

$$f(\beta) = \beta^3 - (1.5\alpha + 2)\beta^2 + (1.5\alpha + 1)\beta - \alpha = 0.$$

Plotting $f(\beta)$ vs. β it is observed that the cubic equation has three roots $0 \leq \beta_1 < \beta_2 \leq 1$ and $\beta_3 \geq 1$ for $\alpha \leq \alpha^* = 0.226$, which makes its discriminant $f(\alpha) = \alpha^3 + \frac{4}{5}\alpha^2 + \frac{28}{15}\alpha - \frac{64}{135} < 0$ (http://en.wikipedia.org/wiki/Cubic_equation.), and a single root $\beta_3 > 1$ for $\alpha > \alpha^*$. *The single parameter α determines the level of lock contention for the standard locking and its critical value α^* determines the onset of thrashing.* The smallest root β_1 for $\alpha \leq \alpha^*$ determines $\overline{M}_A = M(1 - \beta_1)$, which increases with M, but drops after a maximum value is achieved. This is because of a snowball effect that blocked txns cause further txn blocking. Given the throughput characteristic $X(n)$, $n \geq 1$ with no lock contention, the throughput of a TP system when $\overline{M}_A$ is non-integer is: $X(\overline{M}_A) = (\lceil \overline{M}_A \rceil - \overline{M}_A) X(\lfloor \overline{M}_A \rfloor) + (\overline{M}_A - \lfloor \overline{M}_A \rfloor) X(\lceil \overline{M}_A \rceil)$. The analysis for txns with variable step durations uses iteration to estimate ρ, which is the ratio of the number locks held by blocked and active txns. It is shown in [12] that the critical value of ρ matches the *conflict ratio* parameter in [17], which is the ratio of the number of locks held by all and active txns.

Restart-Oriented Locking Methods

Performance degradation in standard locking is mainly caused by txn blocking. Txn aborts and restarts to resolve deadlocks have little effect on performance, because deadlocks are rare. Restart-oriented locking methods follow the strict 2PL paradigm, but txns encountering lock conflicts are restarted to increase the degree of txn concurrency, so that a higher txn throughput can be attained. This is beneficial for high lock contention TP systems with excess processing capacity, but in hardware resource bound systems this may result in performance degradation with respect to standard locking. Since active txns are not guaranteed to

complete successfully and commit, txn throughput cannot be expressed as $X(\overline{M}_A)$, as in the case of standard locking. The *Running Priority* (RP) method aborts txn T_B in the waits-for-graph $T_A \rightarrow T_B \rightarrow T_C$, when T_A has a lock conflict with T_B [2]. Similarly T_B which is already blocking T_A is aborted when it is blocked requesting a lock held by T_C. In effect RP increases the degree of txn concurrency even if the restart of an aborted txn is delayed. The no-waiting [8] and cautious waiting methods abort and restart a txn encountering a lock conflict with an active and blocked txn, respectively. There is no benefit, since the restarted txn will encounter the same lock conflict as before. The *Wait Depth Limited* (WDL) method measures txn progress by its length (L), which in [2] is approximated by the number of locks that it holds in a lock contention bound system, while attained CPU time would be a concern otherwise. WDL differs from RP in that it attempts to minimize wasted processing by not restarting txns holding many locks or txns nearing completion. This requires a priori knowledge of txn's locking requirements. For example, if T_A which is blocking another txn has a lock conflict with T_B, then if $L(T_A) < L(T_B)$ then abort T_A, else abort T_B [2]. The superior performance of WDL over RP and especially standard locking has been shown via random-number-driven (Monte-Carlo) simulation in [2], but also trace-driven simulation in [17]. RP and WDL attempt to attain *essential blocking*, i.e., allowing a txn to be blocked only by active txns and not requesting a lock until it is required. Immediately restarting an aborted txn may result in *cyclic restarts* or *livelocks*, which should be prevented to reduce wasted processing and to ensure timely txn completion. *Restart waiting* delays the restart of an aborted txn until *all* conflicting txns are completed [2]. Conflict avoidance delays with random duration are a less reliable method to prevent cyclic restarts.

Restart-oriented methods are analyzed using a Markov chain model in [11], while flow-diagrams are utilized in the analysis of the no-waiting method in [8]. Active (resp. blocked) states correspond to S_{2j}, $0 \le j \le k$ (resp. S_{2j+1}, $0 \le j \le k-1$). For example, in the case of the no-waiting policy there is a transition forward when there is no lock conflict: $S_{2j} \rightarrow S_{2j+2}$, $0 \le j \le k-1$, otherwise the txn is aborted: $S_{2j} \rightarrow S_0$. The state equilibrium equations for the Markov chain yield the steady-state probabilities π_i, $0 \le i \le 2k$. The mean number of visits v_i to S_i can be similarly obtained by noting that $v_{2k} = 1$, which is the state at which the txn commits. Given that h_i denotes the mean holding time at S_i, then $\pi_i = v_i h_i / \sum_{j=0}^{2k} v_j h_j$, $0 \le i \le 2k$. The mean number of txn restarts is given by $v_0 - 1$. A shortcoming of the analytic approach is that requested locks are implicitly resampled.

Optimistic Concurrency Control

A txn starting its execution first copies all objects required for its processing into its private workspace [6]. To ensure serializability after completing its processing, the txn undergoes validation to ascertain that none of the copied objects was modified by another txn after it was read. If validation is successful the txn commits and otherwise it is immediately restarted. Given that the k objects updated by txns are uniformly distributed over a database of size D, the probability of data conflict between two txns of size k is: $\psi = 1 - (1 - k/D)^k \approx k^2/D$. The completion rate of successfully validated txns is $p\mu s(M)$, where p is the probability of successful validation, μ is the completion rate of txns, and $s(M)$ takes into account the hardware resource contention due to multiprogramming. A txn with processing time x will require $xM/s(M)$ time units to complete. It observes the commits of other txns as a Poisson process with rate $\lambda = (1 - 1/M)p\mu s(M)$. Txns encountering data conflict with rate $\gamma = \lambda\psi$, fail their validation with probability $q = 1 - e^{-\gamma xM/s(M)}$. The number of txn executions in the system follows a geometric distribution $P_j = q(1-q)^{j-1}$, $j \ge 1$ with a mean $\overline{J} = 1/q = e^{\gamma xM/s(M)}$. The system efficiency p can be expressed as the ratio of the mean execution time of a txn without and with data contention, i.e., with the possibility of restarts due to failed validation:

$$p = \frac{E[xM/s(M)]}{E[(xM/s(M))e^{\gamma xM/s(M)}]} = \frac{\int_0^\infty xe^{-\mu x}dx}{\int_0^\infty xe^{(\gamma M/s(M)-\mu)x}dx} = [1-(M-1)\psi p)]^2.$$

The equation yields one acceptable root $p = [1+2(M-1)\psi - \sqrt{1+4(M-1)\psi}]/[2(M-1)^2\psi^2] < 1$, provided $\gamma M/s(M) < \mu$ and the integral in the denominator converges. The txn execution time is not resampled in this analysis, because resampling processing times results in a mix of completed txns which is shorter than original. Shorter txns have a higher probability of successful validation and resampling results in overestimating performance. This analysis in [6] is that of

the OCC *silent* [7] or *die* method [2], which is inefficient in that a conflicted txn is allowed to execute to the end, at which point it is restarted. The analysis in [6] also considers *static* data access, i.e., all objects are accessed at the beginning of execution, while *dynamic* or *on-demand* data access is more realistic. The OCC *broadcast* or *kill* method aborts a txn as soon as a conflict is detected. The analysis of the static/silent method in [6] is extended to the other three cases in [7]. The distribution of txn processing time affects performance and in a system with multiple txn sizes the wasted processing is mainly due to lengthy txns [7]. This is due the *quadratic effect* that the probability that a txn encounters a data conflict increases with the number of objects it accesses (k) and its processing time, which is also proportional to k [2]. While the optimistic die method seems to be less efficient than the optimistic kill method, this is not the case when due to buffer misses txns make disk accesses. It is advantageous then to allow a conflicted txn to run to its completion to prime the database buffer with all the objects required for its re-execution in a second processing phase. In *two-phase processing* txns in the second execution phase have a much shorter processing time, since disk accesses are not required, provided *access invariance* prevails, i.e., a restarted txn accesses the same set of objects as it did in its first execution phase [2]. The second execution phase may use the optimistic kill method or even lock preclaiming which ensures that the txn will not be restarted. In general we have multiphase processing methods with different CC methods with increasing strengths, e.g., locking vs. OCC. WDL and RP restart-oriented locking methods outperform two-phase methods [2]. The analysis in [7] can be extended to take into account the variability in txn processing times and the use of different CC methods across txn phases.

Conclusion

There is a need for methodologies combining analytic models, simulation, and trace analysis to explore the effectiveness of new computer organizations for TP, since it is expensive to build realistic prototypes for experimentation. An abstract model of a standard locking system is analyzed here to provide an understanding of the thrashing phenomenon in standard locking. It is shown that a single parameter α determines the level of lock contention with a critical value, which determines the onset of thrashing. Restart-oriented locking methods selectively abort txns to increase the degree of txn concurrency and reduce the level of lock contention, by disallowing txns to wait for the completion of already blocked txns. Two-phase processing methods reduce the data contention level in TP systems with access invariance, by lowering the holding time of database objects. Both methods require excess processing capacity to cope with the additional processing when txns are restarted.

Cross-references

- ▶ Optimistic Concurrency Control
- ▶ RAID
- ▶ Strict 2PL
- ▶ Two-Phase Locking

Recommended Reading

1. Chen P.M., Lee E.K., Gibson G.A., Katz R.H., and Patterson D.A. RAID: High-performance, reliable secondary storage. ACM Comput. Surv., 26(2):145–185, 1994.
2. Franaszek P., Robinson J.T., and Thomasian A. Concurrency control for high contention environments. ACM Trans. Database Syst., 17(2):304–345, 1992.
3. Gray J.N. and Reuter A. Transaction Processing: Concepts and Facilities. Morgan Kauffmann, Los Altos, CA, 1992.
4. Kleinrock L. Queueing Systems: Vol. 1/2: Theory/Computer Applications. Wiley, New York, 1975/1976.
5. Lazowska E.D., Zahorjan J., Graham G.S., and Sevcik K.C. Quantitative System Performance. Prentice-Hall, Englewood Cliffs, NJ, 1984.
6. Morris R.J.T. and Wong W.S. Performance analysis of locking and optimistic concurrency control algorithms. Perf. Eval., 5(2):105–118, 1985.
7. Ryu I.K. and Thomasian A. Performance evaluation of centralized databases with optimistic concurrency control. Perf. Eval., 7(3):195–211, 1987.
8. Tay Y.C. Locking Performance in Centralized Databases. Academic Press, New York, 1987.
9. Tay Y.C. and Zou M. A page fault equation for modeling the effect of memory size. Perf. Eval., 63:99–130, 2006.
10. Thomasian A. Concurrency control: Methods, performance, and analysis. ACM Comput. Surv., 30(1):70–119, 1998.
11. Thomasian A. Performance analysis of locking policies with limited wait-depth. Perf. Eval., 33(1):1–21, 1998.
12. Thomasian A. Two-phase locking and its thrashing behavior. ACM Trans. Database Syst., 18(4):579–625, 1993.
13. Thomasian A. and Blaum M. Mirrored disk reliability and performance. IEEE Trans. Comput., 55(12):1640–1644, 2006.
14. Thomasian A., Fu G., and Han C. Performance evaluation of two-disk failure tolerant arrays. IEEE Trans. Comput., 56(6):799–814, 2007.

15. Thomasian A. and Menon J. RAID5 performance with distributed sparing. IEEE Trans. Parallel Distr. Syst., 8(6):640–657, 1997.
16. Thomasian A. and Ryu I.K. Performance analysis of two-phase locking. IEEE Trans. Software Eng., 17(5):386–402, 1991.
17. Weikum G., Hasse C., Moenkeberg A., and Zabback P. The COMFORT automatic tuning project. Inf. Syst., 19(5):381–432, 1994.

Performance Benchmark

► Application Benchmark

Performance Measures

► Search Engine Metrics

Performance Metrics

► Evaluation Metrics for Structured Text Retrieval

Performance Monitoring Tools

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

Definition

Performance monitoring tools denote the utilities and programs that give access to database server internals.

Historical Background

Relational database systems have had to prove their performance from the outset [1]. Tools to measure their performance have therefore been present in early versions of most major systems.

Foundations

Performance Monitoring tools are useful in finding out how queries are being serviced, and how the underlying resources are being used. In this entry, the characteristics of the most relevant types of tools are described.

Event Monitor

Event monitors capture the aggregate resources associated to a given event (e.g., query execution, deadlock, session) and report the collected data when the event completes. Event monitors should have low overhead as the collected data is usually accumulated in performance counters that are updated as a side effect of the operations monitored (e.g., CPU usage, IO issued, locks collected during query execution) and only accessed once the event has completed.

Event monitors are useful to identify the queries that consume most resources and should require the attention of the database tuner.

Query Plan Explainer

Query plan explainers display the execution plan chosen by the query optimizer for a given query. Explainers represent a query plan, either in textual form or graphically, as a tree whose leaves are base tables and internal nodes are access methods and relational operators. The nodes are possibly annotated with relevant properties (e.g., cardinality, estimated cost).

Query plan explainers are useful to check that the optimizer relies on updated statistics to generate a query execution plan, to check that indexes are used as intended, or to find out whether costly operators are inserted against the programmer's better judgment.

Profiler

There are two types of profilers:

1. Time-based profilers address the question: *how is the time spent*? A time-based profiler logs the time spent in the different components of the database system and presents the time spent processing as well as the time spent waiting for resources.
2. Counter-based profilers address the question: *how are resources used*? A counter-based profiler uses counters to monitor the resources used during execution and presents either database-wide or system-wide aggregate counters.

Profilers might incur high running cost because of the overhead of logging a potentially large number of actions. Profilers are most useful to understand the behavior of critical queries.

Key Applications

Database management system developers need performance monitoring tools to make sure that their system is performing well. Database administrators need performance monitoring tools to make sure that their database instance is performing as well as possible.

Cross-references

► Query Optimization

Recommended Reading

1. McJones P. The 1995 SQL reunion: people, projects, and politics. Technical Report: SRC–TN–1997–018.
2. Millsap C. and Holt J. Optimizing Oracle Performance. O'Reilly, Sebastopol, CA, 2003.
3. Shasha D. and Bonnet P. Database Tuning: Principles, Experiments and Troubleshooting Techniques. Morgan Kaufmann, San Francisco, CA, 2002.

Period-Stamped Temporal Models

Nikos A. Lorentzos
Agricultural University of Athens, Athens, Greece

Synonyms

Interval-based temporal models

Definition

A period-stamped temporal model is a *temporal data model* for the management of data in which time has the form of a *time period*.

Historical Background

There are applications that require the recording and management not only of data but also of the time during which this data is valid (*valid time*). A typical example is the data maintained by pension and life insurance organizations in order to determine the benefits for which a person qualifies. Similarly, such organizations have to record their financial obligations at various times in the future.

There are also sensitive applications, in which it is important to record not only the data but also the time at which this data was either recorded in the database or deleted or updated (*transaction time*). This time is recorded automatically by the Database Management System (DBMS) and it cannot be modified by the user.

Finally, there are applications in which both data, valid time, and transaction time need be recorded.

A data model for the management of data and also of either valid time or transaction time is a *temporal data model*. If time has the form of a *time period* the model is a *period-stamped data model*.

In the case of the relational model, data and time are recorded in a relation. A relation to record data and valid time is a *valid time* relation. A relation to record data and transaction time is a *transaction time* relation. Finally, a relation to record data, valid time and transaction time is a *bitemporal relation*.

In spite of the interest for the management of temporal data, there are many practical problems, which cannot be supported directly by the use of a conventional DBMS. As a consequence, much programming is required in order to support such applications. Hence, the satisfaction of actual user requirements necessitated the definition of a period-stamped temporal model. The bulk of research, on the definition of such a model, appeared in the 1980's, at a time when computers became powerful enough to process large volumes of data.

Foundations

Problems related to the management of temporal data are depicted next. The illustration is restricted to the relational model, in particular to the management of valid time relations. Note, however, that relevant problems can be identified in the management of transaction time relations whereas the management of bitermporal relations is much more complicated.

Examples Illustrating Inadequacy of Conventional Models to Support Time Period

Figure 1 shows SALARY, a valid time relation that records salary histories. The notation "d_number," for values recorded in attributes Begin and End,

Salary

Name	Amount	Begin	End
Alex	100	d100	d299
Alex	150	d300	d499
Alex	150	d800	d999
John	100	d200	d899

Period-Stamped Temporal Models. Figure 1. Example of a valid time relation.

represents a point in time, which is of a date-time data type. For example, d100 could represent "January 1, 2007." The interpretation of the first tuple is that Alex earned 100 on each of the dates in the *time period* [d100, d299], i.e., on each of the dates d100, d101,..., d299. Then the amount earned was 150 for each of the dates in [d300, d499].

Contrary to the simple representation of a valid time relation, a conventional DBMS lacks the functionality required for the management of such relations. This is illustrated below by a number of examples.

Data insertion: If the tuple (Alex, 150, d400, d799) is inserted into SALARY, it will be recorded in addition to the other tuples. In this case, however, SALARY will contain duplicate data for Alex's payment on the dates d400, d401,...,d499, since this data is already recorded in the second tuple of the relation. Moreover, the query, to return *the time at which Alex's salary became 150*, if not formulated carefully, will return three dates, d300, d400 and d800 whereas the correct answer should be d300. To avoid such problems, it would be appropriate for the newly inserted tuple to be combined with the second and third tuple for Alex into a single one, (Alex, 150, d300, d999), i.e., tuples with identical data for attributes Name and Amount (*value equivalence*), which have either *overlapping* or *adjacent* time periods, to coalesce into a single time period (*temporal coalescing*).

Key: Declaring the primary key of SALARY, to be a multi-attribute key of Name and Amount, will not disallow the recording of the tuple (Alex, 200, d400, d799). As a result, SALARY will contain conflicting data, since its second tuple will be showing that Alex's payment is 150 for each of the dates d400, d401,...,d499 whereas, from the newly inserted tuple, it will also be shown that, for these dates, this payment is 200. Hence, in the case of a conventional DBMS, special integrity constraints have to be defined for every relation like SALARY.

Data deletion: Assuming that John's payment for the dates d400, d401,...,d799 was recorded by mistake and has to be deleted, the last tuple of SALARY has to be replaced by two tuples, (John, 100, d200, d399) and (John, 100, d800, d899).

Data update: Assuming that John's payment for dates d400, d401,...,d799 has to be corrected to be 150, the last tuple of SALARY has to be replaced by three tuples, (John, 100, d200, d399), (John, 150, d400, d799) and (John, 100, d800, d899).

Data projection: The query "*for every employee, list the time during which he is paid*," requires a projection on attributes Name, Begin, End. This will generate four tuples. In practice, however, a *temporal coalescing* of the time periods recorded in SALARY is also required in order to obtain the correct result, consisting of only three tuples, (Alex, d100, d499), (Alex, d800, d999) and (John, d200, d899).

Since a conventional DBMS lacks the functionality illustrated above, special code has to be written to handle correctly all the cases described. This limitation gave rise to research on the definition of a temporal data model. Many of the research efforts led to proposals for the definition of various period-stamped data models.

Common Characteristics of Period-Stamped Data Models

The majority of researchers adopted the principle that a temporal data model should be a minimal extension of the conventional relational data model. The most common characteristics of these approaches are outlined below.

A period-stamped, valid time relation always has two types of attributes: The first type consists of one or more ordinary attributes of a conventional relation. These attributes are termed *explicit* by some authors. Valid time itself is not considered to be data, it is considered as being *orthogonal* to data. Hence, the second type consists of special attributes to record valid time. These attributes are often termed *implicit* and can be system-assigned by default. Some approaches consider one pair of such attributes, to record the Begin and End of valid time periods. The valid time recorded in them is usually assumed to represent a time period of the form [Begin, End], i.e., an interval closed on both sides. It is said that the data recorded in the explicit attributes is *stamped* by the time recorded in the implicit attributes.

According to the concepts discussed above, the explicit attributes of the relation in Fig. 1 are Name and Amount and the implicit attributes are Begin and End. The first tuple of the relation is (Alex, 100) and it is *stamped* by the time period [d100, d299].

Some approaches define a *time period* data type; therefore they consider only one implicit attribute. In such approaches, the equivalent scheme for the relation in Fig. 1 is SALARY (Name, Amount, Period).

Since valid time is not considered to be data, it cannot be part of the primary key of a relation. Hence, Name is designated as the primary key of SALARY, i.e., it matches the primary key of an ordinary relation.

In all the approaches, valid time is considered to be discrete. Hence, the time period [d100, d299] is interpreted as consisting of the dates d100, d101,...,d299. Various *temporal granularities* can be declared by the user, depending on the application.

Temporal Algebras, *Temporal Query Languages* or Temporal Calculi are proposed in various approaches. The functionality of the operations of the conventional relational model is revised accordingly, so as to overcome the problems illustrated earlier. Appropriate predicates are also defined, applicable to time periods, which can be used in selection operations. In some approaches additional operations are defined to capture temporal functionality that cannot be achieved by simply extending existing languages. In general, an implicit *temporal coalescing* takes place in all the temporal operations introduced.

Most of the proposed models in the literature support the valid time property over past, present and future time periods.

Desired Behavior of Period-Stamped Data Models

Given the above limitations, new problems have to be faced by the proposed period-stamped models. The most interesting of them are illustrated next by making use of relation SALARY in Fig. 1.

Data projection: The query "*list all the employees ever paid*" requires a projection of SALARY on Name. Such an operation normally yields a relation R with tuples (Alex), (John). Given however that R lacks implicit attributes, it gives rise to the question whether it is an appropriate response. To overcome this problem, in some approaches projection is defined to yield, after a *temporal coalescing*, the rows (Alex, d100, d499), (Alex, d800, d999) and (John, d200, d899). Notice however that, in this case, the result matches exactly that of another query, "*for every employee, list the time during which he is paid.*"

Stamp projection: Since time is not data, some special operation, to project only on time, may be necessary. For example, the answer to the query "*give the time during which at least one employee was paid*" should consist of one row, (d100, d999). Given however that this result lacks explicit attributes, it is necessary to determine whether it is appropriate to allow time stamp projection without explicit attributes. Similar questions arise in the case of projections of only one of the two implicit attributes.

Association of data in distinct relations: Such an association requires the use of the Cartesian product operation. For an illustration, consider also the relation in Fig. 2, which is used to record the history of employee assignments to departments. Then a Cartesian product of SALARY with ASSIGNMENT seems to yield a relation with two pairs of Begin and End time stamps, one pair from each relation. Clearly this is not a correct period-stamped relation. The same is also true for a join operation. As a consequence, some researchers avoid defining Cartesian product and restrict to the definition of a *temporal join* operation. This operation yields relation R, in Fig. 3, with a single pair of implicit attributes.

Literature Overview

The characteristics of individual approaches are outlined below. Unless otherwise specified, a point-stamped valid time relation, in the approach under discussion, is that of Fig. 1.

Jones and Mason [1] define LEGOL, a language for the management of period-stamped, valid time relations. It is an early, yet incomplete piece of work.

Ben-Zvi defines a model for the management of period-stamped, bitermporal relations ([14], Chap. 8). One more attribute is used, to record the time at which a tuple is deleted. Sets of time periods are also

Assignment

Name	Department	Begin	End
Alex	Shoe	d100	d199
Alex	Food	d300	d800
John	Food	d200	d899

Period-Stamped Temporal Models. Figure 2. Another valid time relation.

R

Name	Amount	Department	Begin	End
Alex	100	Shoe	d100	d199
Alex	150	Food	d300	d499
Alex	150	Food	d800	d800
John	100	Food	d200	d899

Period-Stamped Temporal Models. Figure 3. Result of a temporal join operation.

considered, consisting of mutually disjoint periods. Some SQL extension is provided, too. It is also an early and incomplete piece of work.

Snodgrass defines a QUEL extension, for the management of valid time, transaction time and bitemporal relations [9]. Data may alternatively be stamped by time points (*point-stamped temporal models*). Time may not be specified for the primary key. Formal semantics are provided. A tuple calculus and a relational algebra are defined in [14], Chap. 6.

Navathe and Ahmed propose an SQL extension for period-stamped valid time relations ([14], Chap. 4, [5,6]). In this approach, the primary key of the relation in Fig. 1 is <Name, Begin>. The definition of the primary key of a relation, in conjunction with the definition of a *Time Normal Form*, enables temporal coalescing. One variation of the select operation takes as argument a time period value. A *moving window* operation enables a temporary *split* of the time stamps of all the tuples into time periods of a fixed size, and the subsequent application of aggregate functions on the data that are associated with these fixed size periods.

Lorentzos and Johnson define a relational algebra for the management of either period-stamped or point-stamped valid time data [3,2]. It is also shown that there are practical considerations for having relations with more than one valid time. A period data type is later defined in [14], Chap. 3. A generic period data type is defined in [14], Chap. 3, and in [4], enabling the use of periods of numbers and strings. Such periods can be used for the management of non-valid time relations which, however, require a functionality identical with that of period-stamped, valid time relations. The operations of the conventional relational model are not revised but two new algebraic operations are defined. Time is treated as data. As such, it may participate in the primary key [4]. In [14], Chap. 3, and in [4], it is shown that there are applications in which a *temporal coalescing* should be disallowed. An SQL extension is also defined in [4].

Sarda defines a relational algebra in [7] and an SQL extension for valid time relations in [14], Chap. 5, and in [8]. A time period data type is also defined in [7]. Data may alternatively be stamped by time points. Time may not be specified for the primary key. The operations of the conventional relational model are not revised but, as reported in [14], if time is projected out, the result relation is not a correct valid time relation. Similarly, the result returned by the Cartesian product operation is considered not to be a correct period-stamped relation. Two additional relational algebra operations are also defined, functionally equivalent with those defined by Lorentzos.

TSQL2 is a consensus temporal SQL2 extension for the management of either period or point-stamped valid time relations, transaction time relations and bitermporal relations. It is complemented by the definition of a relational algebra.

All the previous approaches incorporate time at the tuple level (*tuple time-stamping*). Contrary to these, Tansel incorporates time at the attribute level (*attribute time-stamping*) [10]. Hence, to record the history of salaries and of assignment to departments, it suffices to consider a relation like that shown in Fig. 4, in place of the two distinct relations in Figs. 1 and 2. The relation consists of two tuples, one for Alex and another for John. Time periods are open for the End time. One exception is the case where the end point of a time period equals *now*, in which case the period is closed (see for example Fig. 4, the assignment of Alex to the Food department). Therefore, data valid in the future is not supported.

The approach considers four types of attributes, *atomic* (Name, in Fig. 4), *set valued* (e.g., to record data such as {Alex, Tom}), *triplet valued* (e.g., to record data such as <[d100, d300), 100>) and *set triplet valued* (Salary and Department, in Fig. 4). The operations of the relevant conventional model are revised accordingly. Four additional operations are defined, which enable transformations between relations with different types of attributes. A QUEL extension is defined in [11]. The approach in [10] is extended in [14], Chap. 7, and in [12], to a nested model, incorporating the well-known nest and unnest operations. A Calculus and Algebra are also defined. The approach in [12] is extended in [13], in order to support nested bitemporal relations, in which valid and transaction time are incorporated at the attribute level.

Employee

Name	Salary	Department
Alex	{<[d100, d300), 100> <[d300, d500), 150> <[d800, d1000), 150>}	{<[d100, d200), Shoe> <[d300, now], Food>}
John	{<[d200, d900), 100>}	{<[d200, d900), Food>}

Period-Stamped Temporal Models. Figure 4. A relation in Tansel's approach.

Key Applications

There are many applications that necessitate the management of period-stamped and, more generally, temporal data. Some examples are the following.

Period-stamped, valid time relations can be used by pension and life insurance organizations in order to determine the benefits a person qualifies for. Such relations are also needed for these organizations to record their financial obligations at various times in the future.

Similarly, period-stamped, transaction time relations can be used in sensitive applications, to enable tracing the content of the database and evaluate some decision taken in the past, with respect to the content of the database at that time.

Future Directions

Given the practical interest of period-stamped data, and given the many differences between different approaches, it will be useful to agree on and develop a standard model by an international organization, such as ISO.

Further research is necessary for the management of period-stamped geographical data and for the definition of a nested period-stamped data model.

Cross-references

- ▶ Point-stamped Temporal Models
- ▶ Supporting Transaction Time Databases
- ▶ Temporal Algebras
- ▶ Temporal Database
- ▶ Temporal data models
- ▶ Temporal Logical Models
- ▶ Temporal Object-Oriented Databases
- ▶ Temporal Query Languages

Recommended Reading

1. Jones S. and Mason P.S. Handling the time dimension in a database. In Proc. Int. Conf. Data Bases, 1980, pp. 65–83.
2. Lorentzos N.A. and Johnson R.G. Extending relational algebra to manipulate temporal data. Inf. Sys., 13(3):289–296, 1988.
3. Lorentzos N.A. and Johnson R.G. TRA a model for a temporal relational algebra. In Temporal Aspects in Information Systems, C. Rolland, F. Bodart, M. Leonard (eds.). North-Holland, pp. 203–215, 1988.
4. Lorentzos N.A. and Mitsopoulos Y.G. SQL extension for interval data. IEEE Trans. Knowl. Data Eng., 9(3):480–499, 1997.
5. Navathe S.B. and Ahmed R. A temporal relational model and a query language.Inf. Sci. Int. J., 47(2):147–175, 1989.
6. Navathe S.B. and Ahmed R. TSQL: a language interface for history databases. In Temporal Aspects in Information Systems, C. Rolland, F. Bodart, M. Leonard (eds.). North-Holland, pp. 109–122, 1988.
7. Sarda N.L. Algebra and query language for a historical data model. Computer J., 33(1):11–18, 1990.
8. Sarda N.L. Extensions to SQL for historical databases. IEEE Trans. Knowl. Data Eng., 2(2):220–230, 1990.
9. Snodgrass S. The temporal query language TQUEL. ACM Trans. Database Sys., 12(2):247–298, 1987.
10. Tansel A.U. Adding time dimension to relational model and extending relational algebra. Inf. Sys., 11(4):343–355, 1986.
11. Tansel A.U. A historical query language. Inf. Sci., 53(1–2):101–133, 1991.
12. Tansel A.U. Temporal relational data model. IEEE Trans. Knowl. Data Eng., 9(3):464–479, 1997.
13. Tansel A.U. Canan Eren Atay: Nested bitemporal relational algebra. In Proc. 21st Int. Symposium on Computer and Information Sciences, 2006, pp. 622–633.
14. Tansel A., Clifford J., Gadia J., Segev A., and Snodgrass R. (eds.). Temporal databases: theory, design and implementation. Benjamin/Cummings, 1993.

Persistence

- ▶ ACID Properties

Persistent Applications

- ▶ Application Recovery

Persistent Archives

- ▶ Digital Archives and Preservation

Personal Data

- ▶ Individually Identifiable Data

Personalized Interfaces

- ▶ Adaptive Interfaces

Personalized Search

► Personalized Web Search

Personalized Web

► Data Integration in Web Data Extraction System

Personalized Web Search

Ji-Rong Wen[1], Zhicheng Dou[2], Ruihua Song[1]
[1]Microsoft Research Asia, Beijing, China
[2]Nankai University, Tianjin, China

Synonyms

Personalized search

Definition

For a given query, a personalized Web search can provide different search results for different users or organize search results differently for each user, based upon their interests, preferences, and information needs. Personalized web search differs from generic web search, which returns identical research results to all users for identical queries, regardless of varied user interests and information needs.

Historical Background

Web search engines have made enormous contributions to the web and society. They make finding information on the web quick and easy. However, they are far from optimal. A major deficiency of generic search engines is that they follow the "one size fits all" model and are not adaptable to individual users. This is typically shown in cases such as these:

1. Different users have different backgrounds and interests. They may have completely different information needs and goals when providing exactly the same query. For example, a biologist may issue "mouse" to get information about rodents, while programmers may use the same query to find information about computer peripherals. When such a query is issued, generic search engines will return a list of documents on different topics. It takes time for a user to choose which information he/she really wants, and this makes the user feel less satisfied. Queries like "mouse" are usually called ambiguous queries. Statistics has shown that the vast majority of queries are short and ambiguous. Generic web search usually fails to provide optimal results for ambiguous queries.
2. Users are not static. User information needs may change over time. Indeed, users will have different needs at different times based on current circumstances. For example, a user may use "mouse" to find information about rodents when the user is viewing television news about a plague, but would want to find information about computer mouse products when purchasing a new computer. Generic search engines are unable to distinguish between such cases.

Personalized web search is considered a promising solution to address these problems, since it can provide different search results based upon the preferences and information needs of users. It exploits user information and search context in learning to which sense a query refers. Consider the query "mouse" mentioned above: Personalized web search can disambiguate the query by gathering the following user information:

1. The user is a computer programmer, not a biologist.
2. The user has just input a query "keyboard," but not "biology" or "genome." Before entering this query, the user had just viewed a web page with many words related to computer mouse, such as "computing," "input device," and "keyboard."

Foundations

User Profiling

To provide personalized search results to users, personalized web search maintains a user profile for each individual. A user profile stores approximations of user tastes, interests and preferences. It is generated and updated by exploiting user-related information. Such information may include:

1. Demographic and geographical information, including age, gender, education, language, country, address, interest areas, and other information;
2. Search history, including previous queries and clicked documents. User browsing behavior when

viewing a page, such as dwelling time, mouse click, mouse movement, scrolling, printing, and bookmarking, is another important element of user interest.

3. Other user documents, such as bookmarks, favorite web sites, visited pages, and emails. Teevan et al. [15] and Chirita et al. [1] demonstrate that external user data stored in a user client is useful to personalize individual search results.

User information can be specified by the user (explicitly collecting) or can be automatically learnt from a user's historical activities (implicitly collecting). As the vast majority of users are reluctant to provide any explicit feedback on search results and their interests, many works on personalized web search focus on how to automatically learn user preferences without involving any direct user efforts [6,8,9,10,13]. Collected user information is processed and organized as a user profile in a certain structure, depending on the need of personalization algorithm. This can be completed by creating vectors of URLs/domains, keywords, topic categories, tensors, or the like.

A user profile can usually aggregate a user's history information and represent the user's *long-term* interests (information needs). Some work has investigated whether such a long-term user profile is ineffective in some cases. Consider the second case that was described in the historical background section: a user will have different needs at different times based on circumstances. In such situations, personalization based on a user's long-term interests may not provide a satisfying performance, because similar results could be returned. Some work [10] has considered the use of a user's active context to represent *short-term* information needs. Search context is incorporated into the user profile, or is constructed as a separate short-term user model/profile and is used in helping infer a user's information needs.

Personalized Search Based on Content Analysis

Personalized web search can be achieved by checking content similarity between web pages and user profiles.

Some work has represented user interests with topical categories. User's topical interests are either explicitly specified by users themselves, or can be automatically learned by classifying implicit user data. Search results are filtered or re-ranked by checking the similarity of topics between search results and user profiles. In some work [2,8], a user profile is structured as a concept/topic hierarchy. User-issued queries and user-selected snippets/documents are categorized into concept hierarchies that are accumulated to generate a user profile. When the user issues a query, each returned snippet/document is also classified. The documents are re-ranked based upon how well the document categories match user interest profiles. Chirita et al. [2] use the ODP (Open Directory Project, http://www.dmoz.org/) hierarchy to implement personalized search. User favorite topics nodes are manually specified in the ODP hierarchy. Each document is categorized into one or several topic nodes in the same ODP hierarchy. The distances between the user topic nodes and the document topic nodes are then used to re-rank search results.

Some other work uses lists of keywords (bags of words) to represent user interests. In [13], a user profile is built as a vector of distinct terms and is constructed by aggregating past user click history. The cosine similarity between the user profile vector and the feature vector of returned web pages are used to re-rank results. Shen et al. [10] first use language modeling to mine immediate search contextual and implicit feedback information. The approach selects appropriate terms from related preceding queries and corresponding search results to expand the current query. In a query session, the viewed document summaries are used to immediately re-rank documents that have not yet been seen by the user. Teevan et al. [15] and Chirita et al. [1] exploit rich models of user interests, built from both search-related information, and other information about the user. This includes documents and emails the user has read and created. In [6], keywords are associated with categories and thus user profiles are represented by a hierarchical category tree based on keywords categories.

Personalized Web Search Based on Hyperlink Analysis

Most generic web search approaches rank importance of documents based on the linkage structure of the web. An intuitive approach of personalized web search is to adapt these algorithms to compute personalized importance of documents. A large group of these works focuses on personalized PageRank. PageRank, proposed by Page and Brin [7], is a popular link analysis algorithm used in web search. The fundamental motivation underlying PageRank is the recursive notion that important pages are those linked-to by many important pages. This recursive notion can be formalized by the "random surfer" model [7] on

the directed web graph G. A directed edge $<p, q>$ exists in G if page p has a hyperlink to page q. Let $O(p)$ be the outdegree of web page p in G. $O(p)$ is equivalent to number of web pages that linked by page p. Let **A** be the matrix corresponding to the web graph G, where $A_{ij} = 1/O(j)$ if page j links to page i, and $A_{ij} = 0$ otherwise. In the random surfer model, when a surfer visits page p, he/she keeps clicking outlinks at random with probability $(1\text{-}c)$, and jumps to a random web page with probability c. c is called teleportation constraint or damping factor. The PageRank of a page p is defined as the probability that the surfer visited page p. Iterative computation of PageRank is done as the following equation:

$$\boldsymbol{v}^{k+1} = (1 - c)\ \boldsymbol{A}\boldsymbol{v}^{k} + c\boldsymbol{u} \qquad (1)$$

Here, $\boldsymbol{u}$ is defined as a preference vector, where $|\boldsymbol{u}| = 1$ and $u(i)$ denotes the amount of preference for page i when the surfer jumps to a random web page i. The global PageRank vector is computed when there is no particular preference on any pages, i.e., $\boldsymbol{u} = [1/n,...,1/n]^{T}$. By setting variant preference to web pages, a PageRank vector with personalized views of web page importance is generated. It recursively favors pages with high preference, and pages linked by high-preference page. This PageRank vector is called a *personalized PageRank vector* (*PPV*). To accomplish personalized web search, a personalized PageRank is computed for each user based upon the user's preference. For example, web pages in the user's bookmarks are set higher preferences in $\boldsymbol{u}$. Rankings of the user's search results can be biased according to the user's Personalized PageRank vector instead of the global PageRank.

Unfortunately, computing a PageRank vector usually requires multiple scans of the web graph [7], which makes it impossible to carry out online in response to a user query. Furthermore, when a large number of users employ a search engine, it is impossible to compute and store so many personalized PageRank vectors offline. Many later works [4,5] make efforts to reduce the computation and storage cost of personalized PageRank vectors. Jeh and Widom [5] support the concept that a user's preference set is a sub-set of a set of hub pages H, selected as those of greater interest for personalization. For each hub page p in H, setting the preference to 1 for page p and 0 for other pages, the corresponding personalized PageRank vector is called a basis hub vector. The authors decompose each basis hub vector in two parts: hub skeleton vector and partial vector. Hub skeleton vector represents common interrelationships between hub vectors, and is computed offline. Each partial vector for a hub page p represents the part of p's hub vector unique to itself. Partial vector can be computed at construction-time efficiently. Finally, a personalized PageRank vector can be expressed as a linear combination of a set of basis hub vectors, and is computed at query time efficiently. Experiments show that the approach is feasible when size of hub set $> 10^4$.

Haveliwala [4] use personalized PageRank to enable "topic-sensitive" web search. The approach precomputes k personalized PageRank vectors using k topics, e.g., the 16 top level topics of the Open Directory. For each topic i, a preference vector $\boldsymbol{u_i}$ is generated. $(u_i)_j$ represents the confidence that web page j is classified into topic i. A PPV is computed base upon preference vector $\boldsymbol{u_i}$. The k personalized PageRank vectors are combined at query time, using the context of the query to compute the appropriate topic weights. The experiments concluded that the use of personalized PageRank scores can improve web search, but the number of personalized PageRank vectors used was limited due to the computational requirements. In fact, this approach modulates the rankings based on the topic of the query and query context, rather than for truly "personalizing" the rankings to a specific individual. Qiu and Cho [9] develop a method to automatically estimate a user's topic preferences based on Topic-Sensitive PageRank scores of the user's past clicked pages. The topic preferences are then used to bias future search results.

Community-based Personalized Web Search

In most of the above personalized search strategies, each user has a distinct profile and the profile is used to personalize search results for the user. There are also some approaches that personalize search results for the preferences of a community of like-minded users. These approaches are called community-based personalized web search or collaborative web search. In a community-based personalized web search, when a user issues a query, search histories of users who have similar interests to the user are used to filter or re-rank search results. For example, documents that have been selected for the target query or similar queries by the community are re-ranked higher in the results

list. Sugiyama et al. [13] use a modified collaborative filtering algorithm to constructed user profiles to accomplish personalized search. Sun et al. [14] proposed a novel method named CubeSVD to apply personalized web search by analyzing correlations among users, queries, and web pages in clickthrough data. Smyth et al. [12] show that collaborative web search can be efficient in many search scenarios when natural communities of searchers can be identified.

Server-Side and Client-Side Implement

Personalized web search can be implemented on either server side (in the search engine) or client side (in the user's computer or a personalization agent).

For server-side personalization, user profiles are built, updated, and stored on the search engine side. User information is directly incorporated into the ranking process, or is used to help process initial search results. The advantage of this architecture is that the search engine can use all of its resources, for example link structure of the whole web, in its personalization algorithm. Also, the personalization algorithm can be easily adapted without any client efforts. This architecture is adopted by some general search engines such as Google Personalized Search. The disadvantage of this architecture is that it brings high storage and computation costs when millions of users are using the search engine, and it also raises privacy concerns when information about users is stored on the server.

For client-side personalization, user information is collected and stored on the client side (in the user's computer or a personalization agent), usually by installing a client software or plug-in on a user's computer. In client side, not only the user's search behavior but also his contextual activities (e.g., web pages viewed before) and personal information (e.g., emails, documents, and bookmarks) could be incorporated into the user profile. This allows the construction of a much richer user model for personalization. Privacy concerns are also reduced since the user profile is strictly stored and used on the client side. Another benefit is that the overhead in computation and storage for personalization can be distributed among the clients. A main drawback of personalization on the client side is that the personalization algorithm cannot use some knowledge that is only available on the server side (e.g., PageRank score of a result document). Furthermore, due to the limits of network bandwidth, the client can usually only process limited top results.

Challenges of Personalized Search

Despite the attractiveness of personalized search, there is no large-scale use of personalized search services currently. Personalized web search faces several challenges that retard its real-world large-scale applications:

1. Privacy is an issue. Personalized web search, especially server-side implement, requires collecting and aggregating a lot of user information including query and clickthrough history. A user profile can reveal a large amount of private user information, such as hobbies, vocation, income level, and political inclination, which is clearly a serious concern for users [11]. This could make many people nervous and feel afraid to use personalized search engines. A personalized web search will be not well-received until it handles the privacy problem well.
2. It is really hard to infer user information needs accurately. Users are not static. They may randomly search for something which they are not interested in. They even search for other people sometimes. User search histories inevitably contain noise that is irrelevant or even harmful to current search. This may make personalization strategies unstable.
3. Queries should not be handled in the same manner with regard to personalization. Personalized search may have little effect on some queries. Some work [1,2,3] investigates whether current web search ranking might be sufficient for clear/unambiguous queries and thus personalization is unnecessary. Dou et al. [3] reveal that personalized search has little effect on queries with high user selection consistency. A specific personalized search also has different effectiveness for different queries. It even hurts search accuracy under some situations. For example, topical interest-based personalization, which leads to better performance for the query "mouse," is ineffective for the query "free mp3 download." Actually, relevant documents for query "free mp3 download" are mostly classified into the same topic categories and topical interest-based personalization has no way to filter out desired documents. Dou et al. [3] also reveal that topical interest-based personalized search methods are difficult to deploy in a real world search engine. They improve search performance for some queries, but they may hurt search performance for additional queries.

Key Applications

Personalized web search is considered a promising solution to improve the performance of generic web search. Currently, Google and other web search engines are trying to do personalized search.

Experimental Results

Experimental results have shown that personalized web search can indeed improve performance of web search. Detailed experimental results can be found in the corresponding reference for each presented method. Dou et al. [3] propose a personalized web search evaluation framework based upon large-scale query logs.

Cross-references

- ▶ Information Retrieval
- ▶ Privacy
- ▶ Relevance Feedback
- ▶ WEB Information Retrieval Models
- ▶ Web Search Relevance Feedback

Recommended Reading

1. Chirita P.A., Firan C., and Nejdl W. Summarizing local context to personalize global web search. In Proc. Int. Conf. on Information and Knowledge Management, 2006.
2. Chirita P.A., Nejdl W., Paiu R., and Kohlschütter C. Using ODP metadata to personalize search. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 178–185.
3. Dou Z., Song R., and Wen J. A large-scale evaluation and analysis of personalized search strategies. In Proc. 33rd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2007.
4. Haveliwala T.H. Topic-sensitive pagerank. In Proc. 11th Int. World Wide Web Conference, 2002.
5. Jeh G. and Widom J. Scaling personalized web search. In Proc. 12th Int. World Wide Web Conference, 2003, pp. 271–279.
6. Liu F., Yu C., and Meng W. Personalized web search by mapping user queries to categories. In Proc. Int. Conf. on Information and Knowledge Management, 2002, pp. 558–565.
7. Page L., Brin S., Motwani R., and Winograd T. The pagerank citation ranking: bringing order to the web. Technical report, Computer Science Department, Stanford University, 1998.
8. Pretschner A. and Gauch S. Ontology based personalized search. In Proc. 11th IEEE Int. Conf. on Tools with Artificial Intelligence, 1999, pp. 391–398.
9. Qiu F. and Cho J. Automatic identification of user interest for personalized search. In Proc. 15th Int. World Wide Web Conference, 2006, pp. 727–736.
10. Shen X., Tan B., and Zhai C. Implicit user modeling for personalized search. In Proc. Int. Conf. on Information and Knowledge Management, 2005, pp. 824–831.
11. Shen X., Tan B., and Zhai C. Privacy protection in personalized search. SIGIR Forum, 41(1):4–17, 2007.
12. Smyth B., Coyle M., Boydell O., Briggs P., Balfe E., Freyne J., and Bradley K. A live-user evaluation of collaborative web search. In Proc. 19th Int. Joint Conf. on AI, 2005.
13. Sugiyama K., Hatano K., and Yoshikawa M. Adaptive web search based on user profile constructed without any effort from users. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 675–684.
14. Sun J.-T., Zeng H.-J., Liu H., Lu Y., and Chen Z. CubeSVD: a novel approach to personalized web search. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 382–390.
15. Teevan J., Dumais S.T., and Horvitz E. Personalizing search via automated analysis of interests and activities. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 449–456.

Personally Identifiable Data

- ▶ Individually Identifiable Data

Perturbation Techniques

- ▶ Randomization Methods to Ensure Data Privacy

Perusal

- ▶ Browsing

Pessimistic Scheduler

- ▶ Two-Phase Locking

Petri Nets

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven, The Netherlands

Synonyms

Place transition nets; Condition event nets; Colored nets

Definition

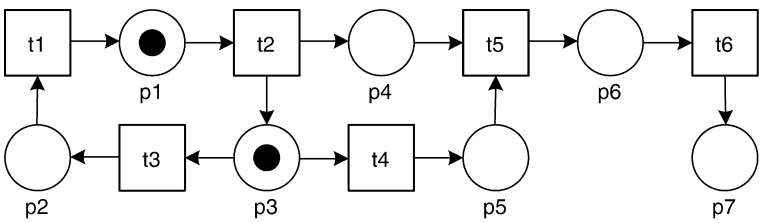
The Petri net formalism provides a graphical but also formal language which is appropriate for modeling systems and processes with concurrency and resource sharing. It was introduced in the beginning of the 1960's by Carl Adam Petri and was the first formalism to adequately describe concurrency. The classical Petri net is a directed bipartite graph with two node types called *places* and *transitions*. The nodes are connected via directed *arcs*. Places are represented by circles and transitions by rectangles. The network structure of the Petri net is static. However, places may contain tokens and the distribution of tokens of places may change as described in the *firing rule*. Petri nets have formal semantics and allow for all kinds of analysis. Moreover, due to the strong theoretical foundation much is known about the properties of different subclasses of Petri nets. Petri nets have been extended in many different application domains, e.g., workflow management systems, flexible manufacturing, embedded systems, communication protocols, web services, asynchronous circuits, etc. Moreover, many modeling languages have been influenced by Petri nets (e.g., UML) and have been mapped onto Petri nets for analysis purposes (e.g., BPEL, BPMN, etc.). The classical Petri net has been extended to allow for the modeling of complex systems, e.g., so called "colors" have been added to model data, timestamps have been added to model time, and hierarchy concepts have been proposed to structure large models. The resulting models are called a *high-level Petri nets*.

Historical Background

Petri started his scientific career with his dissertation "Communication with Automata" [7], which he submitted to the Science Faculty of Darmstadt Technical University in July, 1961. He defended his thesis there in June, 1962 [8]. Petri's dissertation has been quoted frequently, but the Petri net formalism as it is known today emerged later. However, the fundamental idea that asynchronous systems are more powerful than synchronous systems was already present. Petri nets as they are used today first appeared in Petri's talk "Fundamentals on the description of discrete processes" at the Third Colloquium on Automata Theory in Hannover in 1965. Using a simple graphical representation consisting of two types of nodes (places and transitions), Petri was able to describe processes exhibiting concurrency and possibly infinitely many states. Figure 1 shows a simple Petri net consisting of six transitions and seven places. The black dots in places *p1* and *p3* denote tokens and are used to represent the initial state. Despite the addition of concurrency, various properties are decidable for Petri nets which are not decidable for Turing machines (e.g., reachability, liveness, boundedness, etc.).

Initially the focus was on understanding concurrency through Petri nets and little emphasis was put on practical applications. This changed over time and Petri nets were used more and more in all kinds of applications. A nice example is the work on office information systems in the late seventies and early eighties. People like Skip Ellis, Anatol Holt, and Michael Zisman worked on information systems which were driven by explicit process models. It is interesting to see that the three pioneers in this area independently used Petri-net variants to model office procedures. The ideas of these people resulted in all kinds of workflow management systems and today most of the workflow management systems use a notation close to Petri nets [1]. Petri nets have been used in many other application domains ranging from embedded systems and asynchronous circuits to flexible manufacturing, communication protocols, and web services. Petri nets also influenced the development of many languages ranging from process calculi to more application oriented languages such as UML, EPCs, etc.

The classical Petri net allows for the modeling of states, events, conditions, synchronization, parallelism, choice, and iteration. However, Petri nets describing real processes tend to be complex and extremely large.



Petri Nets. Figure 1. A Petri nets consisting of six transitions and seven places.

Moreover, the classical Petri net does not allow for the modeling of data and time. To solve these problems, many extensions have been proposed. Three well-known extensions of the basic Petri net model are: (1) the extension with *color* to model data, (2) the extension with *time*, and (3) the extension with *hierarchy* to structure large models. A Petri net extended with color, time, and hierarchy is called a *high-level Petri net* [2]. Using high-level Petri nets it is much easier to describe complex systems and processes. However, analysis of high-level Petri net other than simulation is often intractable.

Foundations

The classical Petri net is a directed bipartite graph consisting of *places* and *transitions* connected via directed *arcs*. Connections between two nodes of the same type are not allowed. Places are represented by circles and transitions by rectangles as shown in Fig. 1.

A *Petri net* is formally defined by a triple (P, T, F):

1. P is a finite set of *places*,
2. T is a finite set of *transitions* $(P \cap T = \emptyset)$,
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of *arcs* (flow relation)

A place p is called an *input place* of a transition t iff there exists a directed arc from p to t. Place p is called an *output place* of transition t iff there exists a directed arc from t to p. The notation $\bullet t$ is used to denote the set of input places for a transition t. The notations $t\bullet$, $\bullet p$ and $p\bullet$ have similar meanings, e.g., $p\bullet$ is the set of transitions sharing p as an input place. In Fig. 1 transition *t2* has one input place (*p1*) and two output places (*p3* and *p4*).

At any time a place contains zero or more *tokens*, drawn as black dots. The *state*, often referred to as marking, is the distribution of tokens over places, i.e., $M \in P \rightarrow$IN. A state is represented as follows: $1p_1 + 2p_2 + 1p_3 + 0p_4$ is the state with one token in place p_1, two tokens in p_2, one token in p_3 and no tokens in p_4. The notation $p_1 + 2p_2 + p_3$ can also be used represent this state. To compare states a partial ordering is defined. For any two states M_1 and M_2, $M_1 \leq M_2$ iff for all $p \in P$: $M_1(p) \leq M_2(p)$.

The number of tokens may change during the execution of the net. Transitions are the active components in a Petri net: they change the state of the net according to the following *firing rule*:

1. A transition t is said to be *enabled* iff each input place p of t contains at least one token.
2. An enabled transition may *fire*. If transition t fires, then t *consumes* one token from each input place p of t and *produces* one token for each output place p of t.

The initial marking shown in Fig. 1 is *p1* + *p3*. In this marking, *t2*, *t3*, and *t4* are enabled. Firing *t2*, results in the state *2p3* + *p4*. Firing *t3*, results in the state *p1* + *p2*. Firing *t4*, results in the state *p1* + *p5*. Note that firing *t3* disables *t4* and vice versa, i.e., there is a non-deterministic choice between *t3* and *t4*. However, *t2* can fire independently from *t3* and *t4* because it is in parallel.

Given a Petri net (P, T, F) and a state M_1, the following notations are used:

1. $M_1 \xrightarrow{t} M_2$: transition t is enabled in state M_1 and firing t in M_1 results in state M_2
2. $M_1 \rightarrow M_2$: there is a transition t such that $M_1 \xrightarrow{t} M_2$
3. $M_1 \xrightarrow{\sigma} M_n$: the firing sequence $\sigma = t_1 t_2 t_3 \ldots t_{n-1}$ leads from state M_1 to state M_n via a (possibly empty) set of intermediate states $M_2, \ldots M_{n-1}$, i.e., $M_1 \xrightarrow{t_1} M_2 \xrightarrow{t_2} \ldots \xrightarrow{t_{n-1}} M_n$

A state M_n is called *reachable* from M_1 (notation $M_1 \xrightarrow{*} M_n$) iff there is a firing sequence σ such that $M_1 \xrightarrow{\sigma} M_n$. Note that the empty firing sequence is also allowed, i.e., $M_1 \xrightarrow{*} M_1$.

(PN, M) is used to denote a Petri net PN with an initial state M. A state M' is a *reachable state* of (PN, M) iff $M_1 \xrightarrow{*} M'$.

The marked Petri net shown in Fig. 1 has an unbounded number of states. Note that the sequence *t2 t3 t1* can be repeated over and over again. In each cycle an additional token is put in place *p4*.

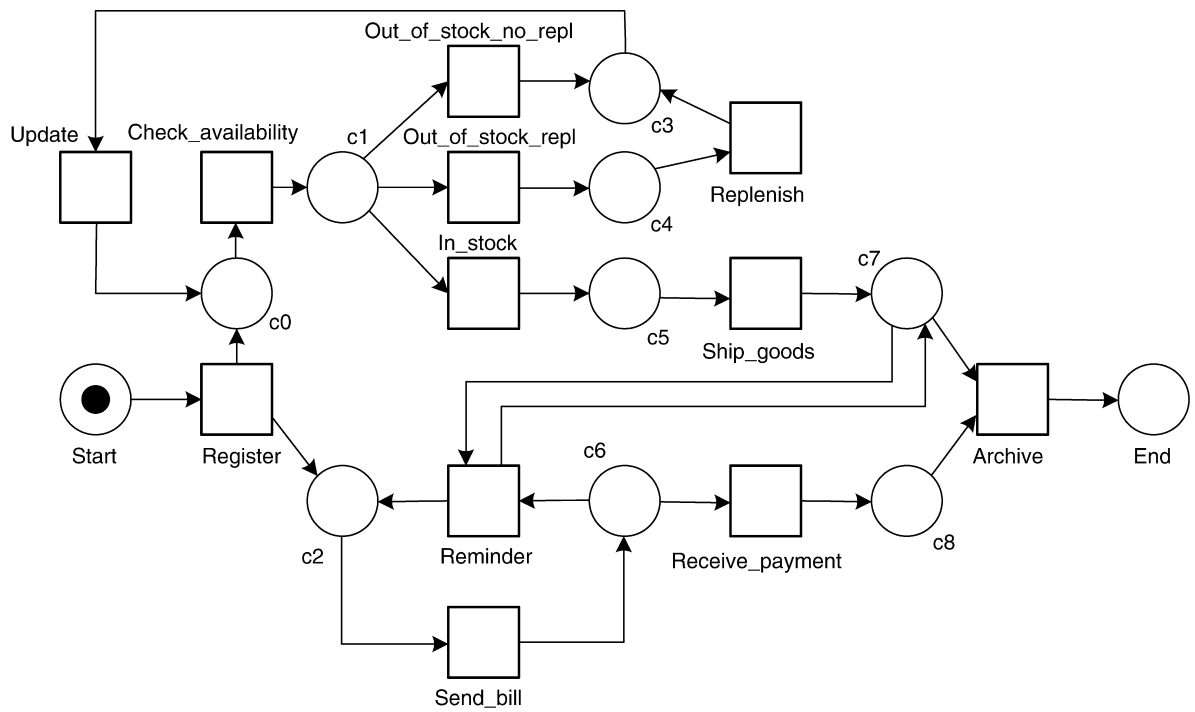
In the remainder, some standard properties for Petri nets are defined [3,4]. First, properties related to the dynamics of a Petri net are defined. Then some structural properties are given.

A Petri net (PN, M) is *live* iff, for every reachable state M' and every transition t there is a state M' reachable from M' which enables t. Note that the marked Petri net shown in Fig. 1 is not live. After firing *t4* twice, it becomes impossible to execute any of the first three transitions. A Petri net is *structurally live* if there exists an initial state such that the net is live. Figure 1 is also not structurally live.

A Petri net (PN, M) is *bounded* iff for each place p there is a natural number n such that for every reachable state the number of tokens in p is less

than *n*. The net is safe iff for each place the maximum number of tokens does not exceed 1. As indicated before, the marked Petri net shown in Fig. 1 is not bounded because the number of tokens in *p4* can exceed any number. A Petri net is *structurally bounded* if the net is bounded for any initially state.

Figure 2 shows a more meaningful example of a Petri net. In fact it models the workflow related to some ordering process and the transitions correspond to tasks. The token in *start* corresponds to an order. Task *register* is represented by a transition bearing the same name. From a routing point of view it acts as a so-called AND-split (two outgoing arcs) and is enabled in the state shown. If a person executes this task, the token is removed from place *start* and two tokens are produced: one for *c0* and one for *c2*. Then, in parallel, two tasks are enabled: *check_availability* and *send_bill*. Depending on the eagerness of the workers executing these two tasks either *check_availability* or *send_bill* is executed first. Suppose *check_availability* is executed first. Based on the outcome of this task a choice is made. This is reflected by the fact that three arcs are leaving *c1*. If the ordered goods are available, they can be shipped, i.e., firing *in_stock* enables task *ship_goods*. If they are not available, either a replenishment order is issued or not. Firing *out_of_stock_repl* enables task *replenish*. Firing *out_of_stock_no_repl* skips task *replenish*. Note that *check_availability*, place *c1* and the three transitions *in_stock*, *out_of_stock_repl*, and *out_of_ stock_no_repl* together form a so-called OR-split: As a result of this construct one token is produced for either *c3*, *c4*, or *c5*. Suppose that not all ordered goods are available, but the appropriate replenishment orders were already issued. A token is produced for *c3* and task *update* becomes enabled. Suppose that at this point in time task *send_bill* is executed, resulting in the state with a token in *c3* and *c6*. The token in *c6* is input for two tasks. However, only one of these tasks can be executed and in this state only *receive_payment* is enabled. Task *receive_payment* can be executed the moment the payment is received. Task *reminder* is an AND-join/AND-split and is blocked until the bill is sent and the goods have been shipped. However, it is only possible to send a reminder if the goods have actually been shipped. Assume that in the state with a token in *c3* and *c6* task *update* is executed. This task does not require human involvement and is triggered by a message of the warehouse indicating that relevant goods have arrived. Again *check_availability* is enabled. Suppose that this task is executed and the result is



Petri Nets. Figure 2. A workflow expressed in terms of a Petri net.

positive, i.e., the path via *in_stock* is taken. In the resulting state *ship_goods* can be executed. Now there is a token in *c6* and *c7* thus enabling task *reminder*. Executing task *reminder* enables the task *send_bill* for the second time. A new copy of the bill is sent with the appropriate text. It is possible to send several reminders by alternating *reminder* and *send_bill*. However, assume that after the first loop the customer pays resulting in a state with a token in *c7* and *c8*. In this state, the AND-join *archive* is enabled and executing this task results in the final state with a token in *end*.

The marked Petri net shown in Fig. 2 is bounded but not live. In fact the model is safe because there are never two tokens in the same place.

Reachability, liveness, and boundedness can be decided for any classical Petri net. For example, it is possible to construct the so-called coverability graph. However, this may be quite inefficient. Fortunately, more efficient techniques are available for different subclasses of Petri nets. Some of these subclasses are listed below.

A Petri net is a *free-choice Petri net* [5]. iff, for every two transitions t_1 and t_2, $\bullet t_1 \cap \bullet t_2 \neq \emptyset$ implies $\bullet t_1 = \bullet t_2$. Figure 1 is free-choice but Fig. 2 is not. Figure 2 is not free-choice because of the two arcs between *c7* and *reminder*. Many properties of free-choice nets can be exploited in their analysis. For example, the combination of liveness and boundedness can be decided in polynomial time.

A Petri net is *state machine* iff each transition has exactly one input and one output place. A Petri net is *marked graph* iff each place has exactly one input and one output transition. Both state machines and marked graphs are examples of free-choice nets and can be analyzed efficiently.

A Petri net $PN = (P, T, F)$ is a WF-net (Workflow net, [1,6]) if and only if:

1. There is one source place $i \in P$ such that $\bullet i = \emptyset$.
2. There is one sink place $o \in P$ such that $o\bullet = \emptyset$.
3. Every node $x \in P \cup T$ is on a path from i to o.

The class of WF-net has been extensively studied because of its applications in workflow management processes and other case-driven processes. Clearly, Fig. 2 is a WF-net while Fig. 1 is not.

For a WF-net with one token on the source place i it is interesting to know whether the net will always terminate properly with a token in the sink place o. This corresponds to the so-called *soundness property*. Soundness can be expressed in terms of liveness and boundedness. If the sink place o is connected to the source place i using some transition $t*$, the so-called short-circuited net is obtained. The original WF-net is sound if and only if the corresponding short-circuited net is live and bounded. This result can be exploited in the analysis of complex workflows.

Key Applications

Workflow Management

Petri nets are used for the modeling, analysis, and enactment of workflows. Many workflow languages have a graphical representation close to Petri nets. Moreover, today's workflow engines use mechanisms comparable to using tokens and the firing rule.

Discrete/Flexible Manufacturing

Traditionally, there have been may applications of Petri nets in manufacturing. The models are used to analyze the manufacturing process in detail, e.g., to find deadlocks or to analyze the performance.

Communication Protocols

Different protocols have been modeled using Petri nets. Various errors have been discovered by verifying established protocols using Petri-net-based analysis techniques.

Embedded Systems

The interactions between hardware and software in embedded systems (e.g., copiers, cars, mobile phones, etc.) may result in all kinds of problems. Therefore, Petri nets are used to model and analyze such systems.

Cross-references

- ▶ Business Process Management
- ▶ Process Mining
- ▶ Web Services
- ▶ Workflow Model Analysis
- ▶ Workflow Patterns

Recommended Reading

1. Brauer W. and Reisig W. Carl Adam Petri and Petri Nets. Informatik-Spektrum, 29(5):369–374, 1996.
2. Desel J. and Esparza J. Free choice Petri nets, volume 40 of Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, Cambridge, UK, 1995.

3. Jensen K., Kristensen L.M., and Wells L. Coloured Petri nets and CPN tools for modelling and validation of concurrent systems. Int. J. Softw. Tools Technol. Trans., 9(3–4):213–254, 2007.
4. Murata T. Petri nets: properties, analysis and applications. Proc. IEEE, 77(4):541–580, April 1989.
5. Petri C.A. Kommunikation mit Automaten. PhD Thesis, Fakultät für Mathematik und Physik, Technische Hochschule Darmstadt, Darmstadt, Germany, 1962.
6. Reisig W. and Rozenberg G. editors. Lectures on Petri nets I: basic models, Springer-Verlag, Berlin Heidelberg New York, 1998.
7. van der Aalst W.M.P. The application of Petri nets to workflow management. J. Circuit. Syst. Comput., 8(1):21–66, 1998.
8. van der Aalst W.M.P. and Hee K.M. van Workflow Management: Models, Methods, and Systems. MIT Press, Cambridge, MA, 2004.

Photograph

► Image

Physical Clock

Curtis Dyreson
Utah State University, Logan UT, USA

Synonyms
Clock

Definition
A *physical clock* is a physical process coupled with a method of measuring that process to record the passage of time. For instance, the rotation of the Earth measured in *solar days* is a physical clock. Most physical clocks are based on cyclic processes (such as a celestial rotation). One or more physical clocks are used to establish a *time-line clock* for a temporal database.

Key Points
Every concrete time is a measurement of some physical clock. For instance a wind-up watch provides the time "now" by measuring the rate at which a coiled, wound spring unwinds. A physical clock is limited by the durability and regularity of the underlying physical process that it measures, so to provide a clock for every instant in a time-line for a temporal database several physical clocks might be needed. For instance a clock based on the rotation of the Earth will (likely be) limited as current models predict the Earth will eventually be gravitationally drawn into the Sun. The modifier "physical" distinguishes this kind of clock from other kinds of clocks in an application, in particular a "time-line" clock. Atomic clocks, which are a kind of physical clock, provide the measurements for International Atomic Time (TAI).

Cross-references
► Chronon
► Time-Line Clock
► Time Instant

Recommended Reading
1. Dyreson C.E. and Snodgrass R.T. Timestamp Semantics and Representation. Inf. Syst., 18(3), 1993, pp. 143–166.
2. Dyreson C.E. and Snodgrass R.T. The baseline clock. The TSQL2 temporal query language, Kluwer, pp. 73–92, 1987.
3. Fraser J.T. Time: the familiar stranger. University of Massachusetts Press, 1987.

Physical Database Design for Relational Databases

Sam S Lightstone
IBM Canada Ltd., Markham, ON, Canada

Synonyms
Database design; Database materialization; Database implementation; Table design; Table normalization; Indexing; Clustering; Multidimensional clustering; Range partitioning; Materialized views; Materialized query tables

Definition
Physical database design represents the materialization of a database into an actual system. While logical design can be performed independently of the eventual database platform, many physical database attributes depend on the specifics and semantics of the target DBMS. Physical design is performed in two stages:

1. Conversion of the logical design into table definitions (often performed by an application developer): includes pre-deployment design, table definitions, normalization, primary and foreign key relationships, and basic indexing.

2. Post deployment physical database design (often performed by a database administrator): includes improving performance, reducing I/O, and streamlining administration tasks.

Generally speaking, physical database design covers those aspects of database design that impact the actual structure of the database on disk. Stage 1 is variably referred to in the industry as an aspect of *logical database design* or *physical database design*. It is known as *logical database design* in the sense that it can be designed independent of the data server or the particular DBMS used. It is also often performed by the same people who perform the early requirements building and entity relationship modeling. Conversely, it is also called *physical database design* in the sense that it affects the physical structure of the database and its implementation. For the sake of this entry, the latter assumption is used, and it is therefore included as part of *physical database design*.

Although it is possible to perform logical design independently of the physical platform and knowledge of database software that will eventually be used, many physical database design tasks depend on the specifics and semantics of the target DBMS (software and hardware). The major tasks of physical database design include: table normalization, table denomalization, indexing, clustering, multidimensional clustering (MDC), range partitioning, hash partitioning, shared nothing partitioning (hash), materialized views (MVs), memory allocation, database storage topology, and database storage object allocation.

Historical Background

Physical database design is as old as database research. Specifically, if this question is limited to relational databases, the first systems were prototyped in the early 1970's. The most elementary problems of physical database design are those of table normalization and index selection. The relational model for databases was first proposed in 1970 by E.F Codd at IBM. The first relational databases using SQL and B+-tree was IBM's System R, in 1976 and the INGRES database at the University of California, Berkeley. The B+-tree, the most commonly-used indexing storage structure for user designed indexes, was first described in the paper *Organization and Maintenance of Large Ordered Indices* by Rudolf Bayer and Edward M. McCreight. While System R was a research prototype, commercialization followed with products by IBM (DB2 – DB2 and Informix are trademarks or registered trademark of International Business Machines Corporation in the United States, other countries, or both. Oracle is a trademarks or registered trademark of Oracle Corporation in the United States, other countries, or both. Other company, product, or service names may be trademarks or service marks of others.) and Oracle around 1980. INGRES also became commercial and was followed by POSTGRES which was incorporated into Informix. Prior to these relational systems, other data models for databases had been used, such as IMS which uses a hierarchical storage model and dominated industrial practice between 1967 and 1980. These pre-relational systems also had unique physical database design challenges.

As relational database systems advanced, new techniques were introduced to help improve operational efficiency, and reduce I/O by partitioning, distributing, and/or better indexing data. All of these innovations, while improving the capabilities and potential for databases, expanded the scope of physical database design increasing the number of design choices and therefore the complexity of optimizing database structures. Though the 1980's and 1990's were dominated by the introduction of new physical database design capabilities, it can also be said that the years since have been dominated by efforts to simplify the process through automation, best practices, or deploying pre-configured "database appliances".

Foundations

The vast majority of physical database design tasks have the primary goal of reducing I/O consumption at runtime for query processing. However, to a lesser degree there are physical design aspects that help improve administrative efficiency by reducing the granularity of backup and restore operations (as in the case of range partitioning), or by improving the efficiency of mass insertion or deletion of data (so called "roll-in" and "roll-out" of data) as in the case of range partitioning. Some features can help reduce CPU or network consumption as in the case of memory tuning for heaps associated with these resources (such as the compiled SQL statement cache found in many commercial database server products).

Infrastructure Mechanics

Indexing schemes typically exploit tree-based structures or hash methods to provide fast access to data,

with B+-trees being the dominant strategy. Range partitioning and clustering methods are based on grouping strategies to physically cluster like or similar data nearby on disk in order to reduce I/O. Hash partitioning methods hash data horizontally across storage objects so that processing can be performed on records associated with the different hash partitioning by different CPUs, providing parallel processing efficiencies. MVs are based on the strategy of precomputation, so that results are available either in full or in part when needed without scanning and calculation. Considerable literature has been published on strategies to match the information within the MV at query compilation time to the queries requiring them, which is non trivial except in cases where the incoming query is an exact match to the MV.

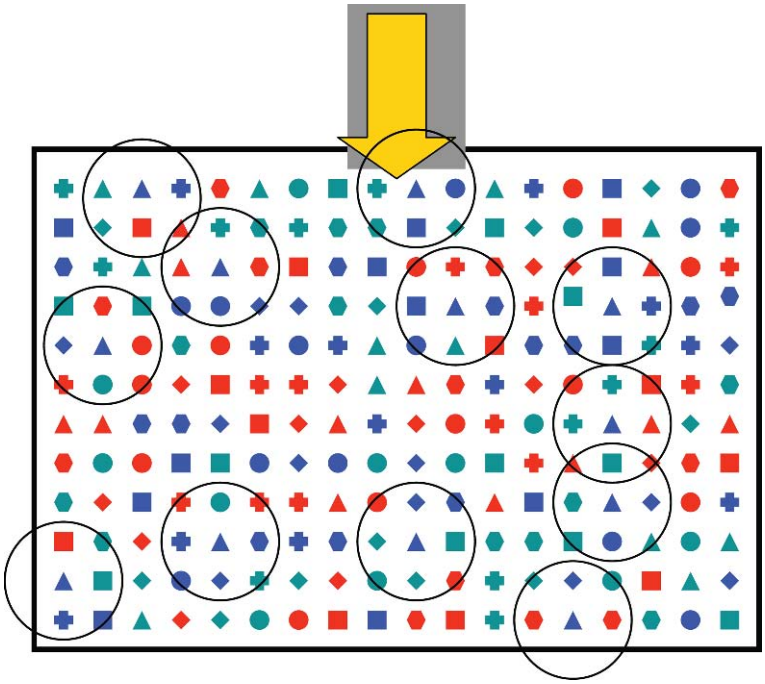
The full breadth of physical database design is beyond the scope of this entry, so this discussion touches only on major aspects.

Design Choices

Physical database design, as performed by end users, has not yet been codified into a scientifically based strategy, and remains as much an art as a science. Typical strategies include cost-benefit analysis of applying various design features. Trial-and-error is still a significant part of the design process. For query processing, indexing is the first design feature considered, and the design process can generally explore the columns used in query predicates within the workload. Range partitioning is generally designed around date fields (by month or by quarter) for roll-in and roll-out processing, and backup granularity. Hash partitioning typically focuses on a unique key or near-unique key in order to efficiently distribute records across CPUs without skew. Data clustering strategies typically focus on columns used in range predicates, or in equality clauses that have high cardinality results sets in order to place similar data that are likely to be jointly required for query resolution physically nearby one another on disk.

Combining Physical Design Choices

Figure 1 shows a conceptual illustration of data stored in one large physical object without any specific treatment of the data, so that the data is unsorted, unclustered, unpartitioned, and not indexed. Data of different characteristics is illustrated by the different colored shapes that appear in the set. One can consider a query that was interested in data represented by the



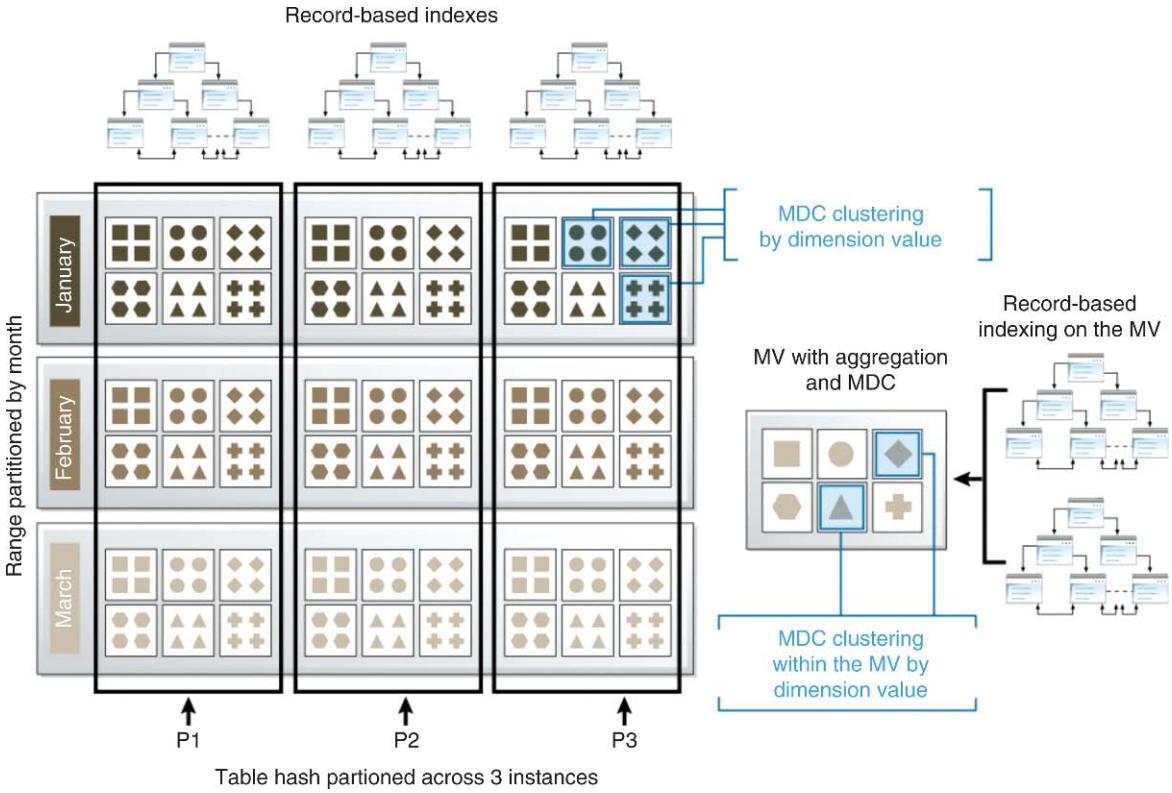
Physical Database Design for Relational Databases. Figure 1. Data of various attributes, stored in a single monolithic table. (image courtesy of K. Beck at IBM)

blue triangles in the image, which appear scattered throughout the object. A full data scan of the set would be required to find and fetch these data.

Figure 2 shows a database with the same data as Fig. 1, which has been carefully designed to exploit a number of physical database design features in combination, such as indexing, range partitioning, MVs, MDC, and hash partitioning. In this new configuration a query for the data represented by the blue triangles is not only clustered in order to perform minimal IO, but the clearly indexed and can be distributed across several CPUs due to hash partitioning. Aggregation on the data can be precomputed and stored in the MV as well.

Other Physical Database Design Techniques

1. Other indexing mechanisms: hash index, bitmap index
   - These indexing strategies use hash functions, or bitmaps to index data that is typically slow changing. Bitmap indexes are favors for data with low cardinality (i.e., few distinct values).
2. Physical design for XML data
   - Semi-structured databases, like XML databases, require alternate indexing strategies.
3. Indexing for Information Retrieval
   - Text indexing is often based on an "inverted tree" index method.
4. Storage and object layout
   - Data placement on disk is affects the efficiency of storage and retrieval. Particularly important when some data has very different frequency or urgency of access than others.
5. Page and block size for I/O efficiency
   - Size of the storage pages, and the blocking levels (number of pages) used during I/O dramatically affect I/O efficiency, where random access favors smaller blocking and while sequential access favors large blocking.
6. Memory configuration and management
   - Complex design choice in the distribution of memory for caching, sorting, hashing, locking, tat typically has large impact on the efficiency of a database, especially if any key memory



Physical Database Design for Relational Databases. Figure 2. Data stored using a number of physical design extensions concurrently.

consumer is constrained. Since memory is a fixed resource, the distribution of memory is a zero sum game (i.e., adding memory for one purpose requires reducing it for another)

7. Distributed database design
   - Distributed databases require complex design choices for data placement and choice of access (particularly where the distributed database includes redundancy, providing choices for which node to access).
8. Shared nothing partitioning
   - Shared nothing partitioning, common in data warehousing, hash partitions data across CPUs (and/or server) to achieve parallel processing over large volumes of data. The choice of hashing key is non trivial. However, in a shared nothing architecture, queries over each hash bucket later need to be shipped and merged since each node (or partition) is assumed to be distinct.
9. Hash partitioned tables
   - Hash partitioning within a server, can divide process of a query across CPUs, while still allowing the efficiencies of shared memory processing within a server.

Key Applications

Lifecycle Differences

It is important to distinguish between physical database design tasks performed during application development, versus those performed following deployment by an administrator.

During application development, the application designer performs logical database design and derives from it a physical design of the database. The physical design assigns typing to the elements of the entities in the database. This phase also designs the database tables, including normalization. Indices are defined, most commonly on primary and foreign keys. Views (non-materialized) are created to abstract the physical implementation away from the application as much as possible and to improve security of the system by controlling access to the data by the logical definition of the views, rather than by the physical structures of the storage.

Following deployment of the database, physical design continues, usually conducted by a database administrator. The design tasks here focus on performance tuning, or performance problem alleviation. Common tasks include the selection of secondary indexes, clustering, MDC, and range partitioning.

A common problem in database application development is that the application designer is forced to develop the database application using small samples of data. DBAs face design challenges when trying to optimize data access for applications that operate over large data sets, which the application designers themselves did not have access to when performing the initial application and database design.

Another key difference between the database design tasks performed at application development time and those following deployment relate to the physical resources of the database server. In most cases the properties of the database server – number and speed of CPUs, CPU-cores, disks, memory, bus bandwidth, etc – are unknown during application development. For applications that are deployed to possibly many sites or many customers there can be many different eventual target servers. Thus physical design qualities of the database that depend on the server's characteristics are inherently the role of the DBA to design. These include data placement and storage topology, memory allocation, and to a lesser degree clustering choices.

The design tasks performed by the DBA occur frequently with the deployment of new applications, application upgrades, or due to the evolution of the data set.

Application Domains

Different application domains require different physical design techniques. The section below highlights the two largest categories of application domains, namely Online Transaction processing (OLTP) and Decision Support Systems (DSS).

OLTP/ERP. Such applications deal with small numbers of records per transaction, and are commonly subject to extremely high concurrency rates. Key physical design characteristics include: (i) Small storage page sizes; (ii) Highly normalized data; (iii) Use of clustering indices or single-dimension MDC to enforce clustering of data; (iv) Memory allocation is heavily used for data caching (bufferpools), and to a lesser degree for caching compiled SQL statements. Memory work area for sorting, hash joins, etc, has considerably reduced demand in this space; (v) Due to the small selectivity of the transactions, the use of range partitioning, MDC with high dimensionality, and view materialization are not heavily used.

Decision Support, Data Warehousing and OLAP. These applications are characterized by large numbers of records accessed for aggregation, cubing etc. Table scans are common, and the workloads often include a high percentage of ad-hoc unpredictable queries, which can not be explicitly optimized for. Key physical design considerations here include: (i) Larger page size and storage blocking to increase the efficiency of scans and prefetching; (ii) Data may be stored in 3NF or Star Schema; (iii) Use of clustering is extremely common, especially long date and geography dimensions; (iv) Range partitioning is heavily used for roll-in and roll-out of data within the warehouse. This is almost always performed along a date dimension (such as by month or by quarter); (v) Memory allocation is complex in these environments. Data caching remains dominant, however, the common occurrence of sorting and hashing in these workloads places increased demand on work memory for these operations. The low concurrency characteristics of these workloads (compared to OLTP systems) place a reduced demand on memory for locking and caching of compiled statements.

Future Directions

The powerful capabilities of modern RDBMSs add complexity and have given rise to an entire professional domain of experts, known as Database Administrators (DBAs), to manage these systems. The problem is not unique to RDBMSs, but is ubiquitous in Information Technology today. Some have called this explosive growth of complexity "creeping featurism". In fact many companies now spend more money recruiting administrators to manage their technology than the money they have spent on the technology itself. One of the key roles of such DBAs is in the development and refinement of physical database design, to achieve "tuning" of the database.

The only real viable long term strategic solution is to develop systems that manage themselves. Such systems are called "self-managing" or "autonomic" systems. Several research efforts have focused on such self-designing database systems, working on problems such as:

1. What-if analysis reusing a database's native query optimizer to explore hypothetical design choices over a real or synthetic workload.
2. Data sampling and statistical analysis to explore the impact of design choices on data distribution and density, or conversely to assess the suitability of target data for design attributes.

Cross-references

- ▶ Advanced Storage Systems
- ▶ Database Design
- ▶ Database Management System
- ▶ Database Middleware
- ▶ Database Security
- ▶ Database Tuning and Performance
- ▶ Distributed Database Systems
- ▶ Hash Partitioning
- ▶ Indexing
- ▶ Multidimensional Clustering
- ▶ Parallel Database
- ▶ Range Partitioning
- ▶ Self-Management Technology in Databases
- ▶ Storage Systems
- ▶ Workflow Management

Recommended Reading

1. Astrahan M.M., Blasgen M.W., Chamberlin D.D., Jim Gray W., King F. I.I., Lindsay B.G., Lorie R.A., Mehl J.W., Price T.G., Putzolu G.R., Schkolnick M., Selinger P.G., Slutz, D.R., Strong H.R., Tiberio, P., Traiger, I.L., Bradford W., Yost W.R.A. System R: A relational data base management system. IEEE Comput, 12(5):42–48, 1979.
2. Bayer R. and McCreight E.M. Organization and Maintenance of Large Ordered Indexes. SIGFIDET Workshop, 107–141, 1970.
3. Finkelstein S., Schikolnick M. and Tiberio P. Physical Database Design for Relational Databases. ACM Trans Database Syst, 13(1):91–128, 1988.
4. Jun R., Chun Zhang, Megiddo N., and Lohman G.M. Automating physical database design in a parallel database. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2002, pp. 558–569.
5. Lightstone S., and Bishwaranjan B. Automated design of Multi-dimensional clustering tables for relational databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp.1170–1181.
6. Lightstone S., Teory T., and Nadeau T. Physical Database Design: The Database Professional's Guide to Exploiting Indexes, Views, Storage, and More. Morgan Kaufmann, 2007.
7. Markos Z., Cochrane R., Lapis G., Hamid P., and Monica U. Answering complex SQL queries using automatic summary tables. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 105–116.
8. Sanjay A., Surajit C., and Narasayya V.R. Automated Selection of Materialized Views and Indexes in SQL Databases, In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 496–505.
9. Sriram P., Bishwaranjan B., Malkemus T., Cranston L., and Huras M. Multi-dimensional clustering: A new data layout scheme in DB2. In Proc. ACM SIGMOD Int. Conf. on Management of Data, pp. 637–641.

10. Surajit C., and Narasayya V.R. AutoAdmin "What-if" index analysis utility. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp.367–378.
11. Surajit C., and Narasayya V.R. Microsoft Index Tuning Wizard for SQL Server 7.0, In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1998, pp.553–554.
12. Teorey T., Lightstone S., and Nadeau T. Database Modeling & Design: Logical Design, 4th edn. Morgan Kaufmann, 2005.
13. Zilio D.C., Jun R., Lightstone S., Lohman G.M., and Storm A.J. Christian Garcia-Arellano, and Scott Fadden. DB2 Design Advisor: Integrated Automatic Physical Database Design. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 1087–1097.
14. Zilio, D.C., Zuzarte C., Lightstone S., Wenbin Ma, Lohman G. M., Cochrane R., Hamid P., Latha S.C., Gryz J., Alton E., Liang D., Valentin G.: Recommending materialized views and indexes with IBM DB2 design advisor. In Proc. 1st Int. Conf. on Autonomic Computing, 2004, pp. 180–188.

Physical Layer Tuning

Philippe Bonnet, Dennis Shasha
University of Copenhagen, Copenhagen, Denmark
New York University, New York, NY, USA

Definition

Tuning the physical layer entails choosing and configuring the underlying hardware and operating system to improve the performance of a database system running a given set of applications.

Historical Background

Ultimately, a database management system runs on processor and memory chips as well as secondary devices such as flash memory and disks. Trying to use those physical resources in the best way possible has been a problem since the first databases were available. Many researchers have noticed that disk capacity has been less of an issue than disk speed (or aggregate bandwidth from an entire disk array). Main memory size has always been a critical factor, because accesses to main memory are so much faster than accesses to disk. Processor speed has been important in applications that mix sophisticated computation with data access (for example in financial analysis), such computations are often handled outside the database server. New technologies have improved the capacity and speed of processors, memory, and disks, but memory is still vastly faster than disk (and somewhat faster than flash memory), so certain tuning principles will remain true.

Foundations

Hardware Tuning

Hardware tuning essentially consists of picking the hardware components on which the database management server is run. The goal is to design a balanced system [2].

Each processing unit consists of one or more processors, one or more disks, and some memory. Assuming a n-GIPS (billion instructions per second) processor, disks will be the bottleneck for on-line transaction-processing applications until the processor is attached to around 30n disks (counting 500,000 instructions per random IO issued by the database system and 70 random IOs per second). Each transaction spends far more time waiting for head movement on disk than in the processor.

Decision-support queries, by contrast, often entail massive scans of a table. In theory, an n-GIPS processor is saturated when connected to n/4 disks (counting 500,000 instruction per sequential IO and 8,000 IO per second, considering 50 MB/second per disk and 64 KB per IO). In practice, the system bus might become the bottleneck. Thus, decision-support sites may need fewer disks per processor than transaction-processing sites for the purposes of matching aggregate disk bandwidth to processor speed. The numbers change when disks are implemented using flash memory, but the principle remains the same.

Operating System Tuning

Tuning the operating system primarily consists of providing an efficient input/output (IO) subsystem to the database server. A database server can issue buffered or unbuffered IO using the file system, or it can issue IO on raw disks thus bypassing the file system. Raw disks offer control and performance to the database system at the cost of a very low level of abstraction. The file system, on the other hand, provides a high level of abstraction, at the cost of decreased performance and double buffering. Further, IO is either synchronous or asynchronous. A synchronous IO forces the thread that issued the issuing thread to block until the IO is complete. Asynchronous IO allows the database server to issue concurrent IO requests on multiple files. Modern operating system such as Linux or Windows, provide database servers with direct asynchronous IO, that provide file abstraction and efficient IO scheduling at a low overhead. Note that out-of-the-box, a

database system will most likely be configured to use standard buffered IO and thus needs to be tuned to use direct asynchronous IO.

As noted above, if one could eliminate the overhead caused by seeks and rotational delay, the aggregate bandwidth could increase by a factor of 10–100. Making this possible requires laying out the data to be read sequentially along disk tracks. Recognizing the advantage of sequential reads on properly laid-out data, most database systems encourage administrators to lay out tables in relatively large *extents* (consecutive portions of disk). Having a few large extents is a good idea for tables that are scanned frequently or (like database recovery logs or history files) are written sequentially. Large extents, then, are a necessary condition for good performance, but not sufficient, particularly for history files. Consider, for example, the scenario in which a database log is laid out on a disk in a few large extents, but another hot table is also on that disk. The accesses to the hot table may entail a seek from the last page of the log; the next access to the log will entail another seek. So, much of the gain of large extents will be lost. For this reason, each log or history file should be the only hot file on its disk, unless the disk makes use of a large RAM cache to buffer the updates to each history file.

When accesses to a file are entirely random (as is the case in on-line transaction processing), seeks cannot be avoided. But placement can still minimize their cost, since seek time is roughly proportional to a constant plus the square root of the seek distance.

The operating system also allows to control the thread model used by the database server and to assign priorities to the different database server threads (and processes). The goal is to favor low latency requests (log writes), ensure fairness among database clients while avoiding starvation. This has several pitfalls however as discussed in [3] below.

Key Applications

A database system depends on the underlying hardware and operating system. Tuning the physical layer is thus a necessary step when deploying any database application.

Cross-references

- ▶ Database Management System
- ▶ Disk
- ▶ Main Memory
- ▶ Processor Cache

Recommended Reading

1. Gray J. and Kukol P. Sequential disk IO tests for GBps land speed record. MS Research Technical Report MSR-TR-2004–62.
2. Gray J. and Shanoy P.J. Rules of thumb in data engineering. In Proc. 18th Int. Conf. on Data Engineering, 2000.
3. Hall C. and Bonnet Ph. Getting priorities straight: improving Linux support for database I/O. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005.
4. Shasha D. and Bonnet P. Database tuning: principles, experiments and troubleshooting techniques. Morgan Kaufmann, 2002.

Physical Time

▶ Transaction Time

Physical Volume

▶ Volume

Physical Window = Tuple-based Windows

▶ Windows

Physician Order Entry

▶ Computerized Physician Order Entry

Pictorial Metadata

▶ Image Metadata

Picture

▶ Icon
▶ Image

Picture Metadata

► Image Metadata

Piecewise-Constant Approximations

► Histograms on Streams

Pipeline

Ryan Johnson
Carnegie Mellon University, Pittsburg, PA, USA

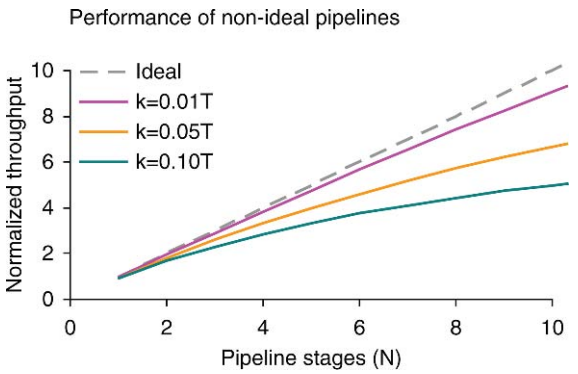
Definition

Pipelining is a performance-enhancing technique which breaks a job into multiple overlapping pieces of work assigned to *pipeline stages*. Each pipeline stage operates in parallel, receiving inputs from a previous stage and passing intermediate results to the next. A pipeline with N stages can accommodate up to N in-progress jobs at a time. A well-designed pipeline requires only a fraction of the resources needed to achieve the same throughput with non-pipelined execution. A *balanced* pipeline, which distributes processing time evenly between all stages, can produce N results during the time it would have taken to process a single job all at once; unbalanced pipelines do not perform nearly as well. Pipelining also provides a secondary benefit of allowing each pipeline stage to specialize for its particular task, potentially making the combined stages both faster and less expensive than the non-pipelined server. Factory assembly lines leverage pipelining to great effect, allowing N specialized workers to achieve far higher throughput than N general purpose workers, while requiring less training per worker and fewer resources overall. The impressive throughput and efficiency pipelining provides make it a useful tool in both hardware and software design. Database systems often improve performance of query processing by executing query plans in pipelined fashion. Pipelining is a form of parallel dataflow execution.

Key Points

Suppose a non-pipelined server requires T time units and R resources to process a job all at once. The server's *latency* (time to process a job start-to-finish) and *occupancy* (time until it can accept more work) are both T. In contrast, a pipelined server with N stages ideally provides latency and occupancy of T' $\leq$ T and T'/N time units, respectively, while consuming R + ε resources. Because peak throughput is the reciprocal of the server's occupancy, pipelined execution potentially provides an N-fold increase in throughput over non-pipelined execution. When plenty of work is available, an initially empty pipeline *fills* or *ramps up* over a period of T' time units to reach peak operating capacity with N in-progress jobs. The pipeline must *stall* any time the first pipeline stage becomes ready and no new jobs are available. Stalls propagate through the pipeline as *bubbles* and reduce throughput by leaving stages idle.

Ideal pipelines assume that (i) the job can be broken into N pieces requiring no more than T/N time units each (possibly less due to gains from specialization), and that (ii) passing work between stages imposes no extra overhead. In practice, pipelines usually have non-negligible stage overhead and tend to be unbalanced, with a *bottleneck stage* that runs slower than the others. Real pipelines therefore have occupancy and latency of T/N + k and T + kN, where k is the total extra delay imposed by imbalance and overhead. Pipeline bubbles, unbalanced stages, and



Pipeline. Figure 1. Impact of unbalanced pipelines and pipelining overhead for as pipeline depth varies. Normalized throughput is equal to 1/(T/N + k)/T. Note that just 1% overhead (and/or imbalance) costs 10% ideal throughput in a 10-stage pipeline.

overhead all become more severe for *deep* pipelines containing many stages, and limit the useful depth of any pipeline. Figure 1 illustrates how even small values of k reduce peak throughput from the ideal as a pipeline gets deeper.

Cross-references

► Dataflow Execution
► Pipelined Query Execution

Pipelined and Independent Parallelism

► Inter-operator Parallelism

Pipelining

Evaggelia Pitoura
University of Ioannina, Ioannina, Greece

Definition

The query execution engine operates on an execution plan produced during query processing which typically is a physical operator graph whose edges specify the dataflow among the operators. There are two basic alternatives for evaluating the execution plan: materialization and pipelining. With materialized evaluation, the results of each operator are created (materialized) and stored in disk and then used as input for the evaluation of the next operator. With pipelining, the results of an operator are passed along as input to the next operator, before the first operator completes its execution. In general, pipelining improves the efficiency of query evaluation by reducing the number of temporary files that are produced.

Key Points

The output of query optimization is an execution plan or operator tree. The execution plan can be thought of as a dataflow graph where the nodes correspond to the physical operators and the edges represent the data flow among these physical operators. The query execution engine is responsible for the execution of this plan. The engine provides generic implementations of all physical operators that take as input one or more data streams and produce one output data stream.

In most database systems, each operator supports an *iterator* interface that allows a parent operator to get the results from its children one tuple at a time. The iterator interface supports *pipelining*, since tuples produced as output by a child operator can be fed as input to their parent in the operator tree, even before the child operator has finished its execution. The alternative would be *materialized evaluation*, in which each operator would fully complete its operation and write its results to disk before the next operator starts execution. By reducing the number of intermediate results written to disk, pipelining improves the efficiency of query evaluation. However, not all physical operators can use pipelining efficiently. For instance, merge-join cannot exploit pipelining in general, since it requires both its input relations to be sorted and it is not possible to sort a relation until all its tuples are available.

Pipelines can be executed in one of two ways: demand-driven or producer-driven. In a *demand-driven* or *demand-pull* pipeline, an operator computes the next tuple or tuples to be returned after receiving a corresponding request from its parent operator. In a *producer-driven* or *push* pipeline, operators do not wait for requests to produce tuples, instead, they generate output tuples eagerly and put them in their output buffers until the buffers become full. Demand-driven pipelining is more common, since it is easier to implement. It also allows for the whole plan to be executed by a single process or thread.

Cross-references

► Execution of Relational Operators
► Iterator
► Parallel Query Execution Algorithms
► Query Processing

Recommended Reading

1. Graefe G. Query evaluation techniques for large databases. ACM Comput. Surv., 25(2): 73–170, 1993.
2. Silberschatz A., Korth H.F., and Sudarshan S. Database System Concepts, 5th edn. McGraw Hill, New York, 2005.

PiT Copy

▶ Point-in-Time Copy

Pixed Oriented Visualiyation Techniques

▶ Dense Pixel Displays

Pixel Classification

▶ Image Segmentation

Place Names

▶ Gazetteers

Place Transition Nets

▶ Petri Nets

Player

▶ Actors/Agents/Roles

Plot

▶ Graph

Plots for Qualitative Information

▶ Visualizing Categorical Data

Point-based Temporal Models

▶ Point-Stamped Temporal Models

Point-based Temporal Data

▶ Atelic Data

Point-in-Time Copy

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Snapshot; PiT copy

Definition

A point-in-time copy is a copy of original data as it appeared at a point in time. In a conventional backup operation, users often create a PiT Copy, while an application is in quiescing, to make the PiT Copy a consistent copy of original data.

Key Points

There are two popular implementation techniques for creating PiT Copies inside a storage system: split mirror and copy on write.

Split mirror is a technique for replicating the original data at a point in time. In some implementations, a storage system replicates the original data, and when users create the PiT Copy, a storage system splits the replication. Copy on Write (CoW) is a technique for capturing data changes to storage and creating a PiT Copy after specifying the point in time. In some implementations, when users create the PiT Copy, a storage system creates its image with both the original data and the modified data.

Techniques for creating PiT Copies have a trade-off between occupied storage capacity and performance. A PiT Copy created by split mirror occupies as much storage as the original data, but a PiT Copy by CoW usually occupies less storage. On the other hand,

CoW usually has more time for accessing the copy source and creating PiT Copies than split mirror.

Cross-references

- ▶ Backup and Restore
- ▶ Logging and Recovery

Recommended Reading

1. Alain Azagury et al. Point-in-time copy: yesterday, today, and tomorrow. In Proc. of IEEE Symp. Conf. on Mass Storage Systems and Technologies, 2002.

Point-Stamped Temporal Models

David Toman
University of Waterloo, Waterloo, ON, Canada

Synonyms

Point-based temporal models

Definition

Point-stamped temporal data models associate database objects with time instants, *indivisible* elements drawn from an underlying time domain. Such an association usually indicates that the information represented by the database object in question is *valid* (i.e., believed to be true by the database) at that particular time instant. The time instant is then called the *timestamp* of the object.

Historical Background

Associating time-dependent data with time instants has been used in sciences, in particular in physics, at least since Isaac Newton's development of classical mechanics: time is commonly modeled as a two-way infinite and continuous linear order of indivisible time instants (often with distance defined as well). Similarly, in many areas of computer science, ranging from protocol specifications to program verification to model checking, discrete point-based timestamps play an essential role. In database systems (and in AI), however, the requirement of *finite* and compact representation has often lead to the use of more complex timestamps, such as intervals. This is particularly common when information about *durations* is stored in a database (in AI, such timestamps are called *fluents*). However, Chomicki [2] has shown that many of these approaches are simply compact representations of large and potentially infinite sets of time instants associated with a particular fact and that, at the conceptual level, the underlying information is often better understood as being timestamped by time instants.

Foundations

Temporal data models are typically defined as extensions of standard non-temporal data models that provide means for representation and manipulation of time-dependent data. Temporal extensions of the relational model accomplish this by relativizing *truth*, i.e., the sets of facts the database believes to be true in the modeled reality, with respect to elements of the time domain. These elements are then called the timestamps of the facts and, intuitively, specify *at which times* the associated facts are true. In the case of point-stamped temporal models, these elements are indivisible time instants drawn from an appropriate time domain.

The main ideas are outlined in a setting in which the time domain is an unbounded countably infinite linear order of *time instants*. Moreover, while the main focus is on *temporal extensions of the relational model*, most of the issues arise in other data models as well, and admit similar solutions.

Timestamps and Database Objects

The choice of timestamps is orthogonal to other choices that define the flavor and properties of the resulting temporal data model, in particular:

1. To the decision of *what database objects the timestamps are attached to*, and
2. To the decision of *how many timestamps* (per such object) are used.

First to explore is the choice of objects to be timestamped. Intuitively, timestamps should only be attached to objects that actually *capture* information in the underlying data model, indicating that the particular (piece of) information is valid for that timestamp. In the relational model these are *tuples* and their membership in *relations*. Thus, the two principal choices are as follows:

The Snapshot Model. Temporal databases in the *snapshot model* are formally defined as mappings from the time domain T to (the class of) standard relational databases with a particular fixed schema ρ. In the

case of a linearly ordered time domain, such a temporal database can be viewed as a time-indexed sequence of standard relational databases, commonly called *snapshots*. Note that such a sequence is not necessarily discrete or finite: that depends on the choice of the underlying time domain. Intuitively, to capture the fact that, in a snapshot temporal database D, a relationship r holds among uninterpreted constants $a_1,...,a_k$ at time t it must be the case that $r(a_1,...,a_k)$ holds in $D(t)$, where $D(t)$ is a standard relational instance, the snapshot of D at time t; this statement is denoted by writing $r^{D(t)}(a_1,...,a_k)$.

These structures, in the area of *modal logics*, are called *Kripke structures* in which worlds are described by relational databases and where the time domain serves as the accessibility relation.

The Timestamp Model. Temporal databases in the *timestamp model* are defined in terms of temporal relations, relations whose schemas are extended with an additional attribute ranging over the time domain. This attribute is commonly referred to as the *timestamp attribute* or simply *timestamp*.

More formally, a relational symbol R is a *timestamped extension* of a symbol $r \in \rho$ if it contains all attributes of r and a single additional attribute t of the temporal sort (without loss of generality, assuming that it is always the first attribute). A *timestamp temporal database* D is then a first-order structure $D = \{R_1^D,...,R_k^D\}$ consisting of the interpretations (instances) for all the temporal extensions R_i of r_i in ρ. The instances R_i^D are called *temporal relations*. Similarly to the snapshot case, there is no restriction on the number of timestamps (i.e., the cardinality of the set of timestamps) in such instances; issues connected with the actual finite representation of these relations are addressed below. However, the relation

$$\{a_1,...,a_k : (t, a_1, ..., a_k) \in R_i^D\},$$

a snapshot of R at t, must be finite for every timestamp $t \in T$.

It is easy to see that snapshot and timestamp temporal models are simply different views of the same (isomorphic) sets of facts and thus represent the same class of temporal databases. Formally, a snapshot temporal database D corresponds to a timestamp temporal database D' (and vice versa) as follows:

$$\forall t.\forall x_1,...,x_k.r^{D(t)}(x_1,...,x_k) \Leftrightarrow R^{D'}(t, x_1,...,x_k),$$

where r and R are a relation in the schema of D and its temporal extension in the schema of D', respectively. This correspondence makes the two models interchangeable. Hence, for temporal queries formulated in query languages such as Temporal Relational Calculus (TRC) or First-order Temporal Logic (FOTL), these two models of point-stamped temporal databases can be used interchangeably.

The Parametric Model. Several temporal data models propose to use time-dependent *unary functions* as attribute values of tuples in temporal relations. These models are called *parametric models* or *attribute timestamped models*. As unary functions can be represented by binary relations (e.g., with the first attribute being the timestamp and the second attribute the value of the function for that timestamp), these models are examples of nested, non-first normal form models [4,5]. Moreover, if the implicit grouping of tuples in such relations is accidental or in the presence of appropriate keys, a first-normal form representation can be obtained by unfolding, similarly to the case of the nested relational model. The transformation is then defined by:

$$R := \{(t, f_1(t),..., f_k(t)) \mid (f_1,...,f_k) \in R^P, t \in T\},$$

where R is a timestamp relation corresponding to the parametric relation R^P. Note also that if a varying number of tuples is to be modeled in this model, *partial functions* must be used in R^P; then, however, tuples can become only partially defined for a given time instant, and therefore additional conditions must be enforced. This makes the model quite cumbersome to use, in particular when compared to the snapshot and timestamp models. Many of these difficulties originate from associating timestamps with *uninterpreted constants* that, in the relational model, *do not* carry any information on their own.

Multiple Atomic Timestamps

The second issue that arises is the question of *how many* timestamps are attached to database objects. So far single-dimensional temporal databases were considered, i.e., where each database object was associated with a single timestamp. The intuition behind such models is that *validity* of data is determined by a single time instant. In the case of the timestamp model, this translates to the fact that temporal relations were allowed

only a single (often *distinguished*) temporal attribute. However, there are two natural reasons to relax this requirement and to allow multiple timestamps to be associated with database objects (i.e., multiple timestamp attributes to appear in schemas of relations, either explicitly or implicitly). There are two cases to consider:

Models with a fixed number of timestamp attributes. In many cases data modeling requires attaching several timestamps to database objects. Intuitively, a particular piece of information can be associated with, e.g., a timestamp for which the information is *valid* in the modeled world and another timestamp that states when the information is *recorded* in the database. Hence, every object has two timestamps. The resulting data model is called the bitemporal model [6] and the timestamps are called valid time and transaction time, respectively. Similarly, one can envision temporal models with three (or any fixed number) of *distinguished temporal attributes* – attributes with a predetermined interpretation – that are *common to all temporal relation schemas*. Temporal data models with an apriori *fixed* number of timestamp attributes can still be equivalently represented using the snapshot and the timestamp approaches. In the snapshot case, database instances are indexed by fixed tuples of time instants. Hence the bitemporal model can be seen as a two-dimensional plane of relational instances. The timestamp model simply adds an appropriate number of (distinguished) attributes to the timestamp extensions of relational schemes.

Models with a varying number of timestamp attributes. While most of the data modeling techniques require only a fixed number of timestamp attributes in schemas of temporal relations, it is often convenient to allow arbitrary number of timestamp attributes to be associated with a database object. This leads to allowing temporal data models without limits on the number of timestamp attributes in schemas of temporal relations. In such models, time dependencies are captured by explicit (user-defined) attributes ranging over the time domain. For a varying number of timestamps, only the *timestamp* view of the temporal data model makes sense.

Fixed-dimensional temporal data models are appealing as they commonly provide an additional built-in *interpretation* for timestamps, e.g., the valid time timestamp always states that the information is *true* in the world at that particular time. Temporal data models with varying, user-defined timestamps do not posses this additional interpretation, and the exact meaning of the timestamps depends on how the world is modeled by the associated attributes (similarly to the standard relational case). However, there is another need for models with varying and unbounded number of timestamp attributes: for temporal query languages based on temporal relational calculus, there cannot be an equivalent temporal relational algebra defined over any of the fixed-dimensional temporal data models (see the entry on Temporal Logic in Database Query Languages or [7,8] for details).

Sets of Timestamps: Compact Representation

The point-stamped temporal data models and the associated temporal query languages provide an excellent vehicle for defining a precise semantics of queries and for studying their properties. However, in *practical applications* an additional hurdle has to be overcome: a naive storage of point-stamped temporal relations, either in the timestamp or in the snapshot model, is often not possible (as the instances of the relations can be infinite, e.g., sets of time instants that represent bounded durations are infinite when a *dense time domain* is used) or impractical (the number of instants is very large). For these and other reasons, many temporal data models associate database objects with *sets* of time instants rather than with single individual time instants.

Temporal models that use complex timestamps, such as intervals, no longer appear to be point-stamped – or in the first normal form (1NF). However, Chomicki [2] has shown that in many cases these complex objects are merely compact representations of a possibly large or even infinite number of time instants associated with a particular fact. The most common approach along these lines is to attach timestamps in the form of *intervals* to facts that persist over time. For example, the temporal relation **WorksFor** in Fig. 1 can be compactly (and finitely) represented by the relation:

$$\mathbf{WorksFor}' = \{([2001, 2002], \text{John}, \text{IBM}), \\ ([2004, \infty], \text{John}, \text{Microsoft})\}.$$

Note that such an *interval encoding* is not unique and multiple snapshot equivalent representations can exist. For single-dimensional temporal relations, uniqueness

Year	Database snapshot (as a set of facts)
...	
1997	{**Degree**(John,BS,MIT)}
1998	{ }
1999	{**Degree**(John,MS,UofT)}
2000	{ }
2001	{**WorksFor**(John,IBM)}
2002	{**WorksFor**(John,IBM)}
2003	{ }
2004	{**Degree**(John,PhD,UW), **WorksFor**(John,Microsoft)}
2005	{**WorksFor**(John,Microsoft)}
...	{**WorksFor**(John,Microsoft)

WorksFor		
Year	Name	Company
200	John	IBM
200	John	IBM
200	John	Microsoft
200	John	Microsoft
..	John	Microsoft

Degree			
Year	Name	Degree	School
1997	John	BS	MIT
1999	John	MS	UofT
2004	John	PhD	UW

Point-Stamped Temporal Models. Figure 1. Instances of matching Snapshot and Timestamp Temporal Database.

of the representation can be achieved using temporal coalescing. However, it is not clear that meaningful, canonical ways of defining coalescing for multi-dimensional temporal relations, including bitemporal relations, exist [3].

On the other hand, the use of temporal coalescing and other set-oriented operations on interval timestamps, such as intersection of timestamps in temporal joins, indicates that the interval timestamps are indeed solely a representation tool for point-stamped relations: these operations cannot be used in a model that associates truth with the intervals themselves as there is no clear way to do so for the intervals that result from such operations, e.g., it is unclear – from a conceptual point of view – what facts should be associated with an intersection of two intervals *without* implicitly assuming that facts associated with the original two intervals hold *for all time points contained in those two intervals*, or at least for certain sub-intervals.

The choice of *intervals* to compactly represent adjacent timestamps originates from the structure of the temporal domain: intervals are the only (convex) one-dimensional sets that can be defined by (first-order) formulas in the language of linear order. For more structured time domains, however, the repertoire of encodings for sets of timestamps can be richer, e.g., using formulas describing *periodic sets*, etc., as compact timestamps.

Query Languages and Integrity Constraints

Point-stamped data models support point-based temporal query languages that are commonly based on extensions of the relational calculus (first-order logic). The two principal extensions are as follows:

1. First-order Temporal Logic: a language with an implicit access to timestamps using temporal connectives and
2. Temporal Relational Calculus: a two-sorted logic with variables and quantifiers explicitly ranging over the time domain.

These languages are the counterparts of the snapshot and timestamp models. However, unlike the data models that are equivalent in their expressiveness, the second language is strictly more expressive than the first [1,8]. Temporal integrity constraints can also be conveniently expressed in these languages [3].

Key Applications

Point-based temporal data models serve as the underlying data model for most abstract temporal query languages.

Cross-references

- ▶ Bitemporal Relation
- ▶ Constraint Databases
- ▶ Data Domain
- ▶ Duplicate Semantics
- ▶ First-Order Temporal Logic
- ▶ Key
- ▶ Nested Transaction Models
- ▶ Non First Normal Form (N1NF)
- ▶ Point-Stamped Temporal Models
- ▶ Relational Model
- ▶ Snapshot Equivalence

- ▶ Temporal Coalescing
- ▶ Temporal Data Models
- ▶ Temporal Element
- ▶ Temporal Granularity
- ▶ Temporal Integrity Constraints
- ▶ Temporal Joins
- ▶ Temporal Logic in Database Query Languages
- ▶ Temporal Query Languages
- ▶ Temporal Relational Calculus
- ▶ Time Domain
- ▶ Time Instant
- ▶ Transaction Time
- ▶ Valid Time

Recommended Reading

1. Abiteboul S., Herr L., and Van den Bussche J. Temporal versus first-order logic to query temporal databases. In Proc. 15th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1996, pp. 49–57.
2. Chomicki J. Temporal query languages: a survey. In Proc. 1st Int. Conf. Temporal Logic, 1994, pp. 506–534.
3. Chomicki J. and Toman D. Temporal Databases. In Handbook of Temporal Reasoning in Artificial Intelligence. M. Fischer, D. Gabbay, and L. Villa (eds.). Elsevier, London, UK, 2005, pp. 429–467.
4. Clifford J., Croker A., and Tuzhilin A. On the completeness of query languages for grouped and ungrouped historical data models. In Temporal Databases: Theory, Design, and Implementation, A. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R.T. Snodgrass (eds.). Benjamin/Cummings, Redwood City, CA, USA, 1993, pp. 496–533.
5. Clifford J., Croker A., and Tuzhilin A. On completeness of historical relational query languages. ACM Trans. Database Syst., 19(1):64–116, 1994.
6. Jensen C.S., Soo M.D., and Snodgrass R.T. Unification of temporal data models. In Proc. 9th Int. Conf. on Data Engineering, 1993.
7. Toman D. On incompleteness of multi-dimensional first-order temporal logics. In Int. Symp. on Temporal Representation and Reasoning and Int. Conf. on Temporal Logic., 2003, pp. 99–106.
8. Toman D. and Niwinski D. First-order queries over temporal databases inexpressible in temporal logic. In Advances in Database Technology, Proc. 5th Int. Conf. on Extending Database Technology, 1996.

Point-versus Period-based Semantics

- ▶ Telic Distinction in Temporal Databases

Polyhedron

- ▶ Simplicial Complex

Polytransactions

GEORGE KARABATIS
University of Maryland, Baltimore County (UMBC), Baltimore, MD, USA

Definition

A polytransaction $T^+$is a transitive closure of a transaction T submitted to an Interdependent Data Management System (a type of a multidatabase system which enforces dependencies among related data objects). The transitive closure is computed with respect to an Interdatabase Dependency Schema (IDS) consisting of a collection of data dependency descriptors (D^3s), which specify the dependencies between data objects.

Key Points

Data objects in multiple (possibly distributed and/or heterogeneous) systems which form dependencies among them are called *interdependent data*. These dependencies are identified through *data dependency descriptors* (D^3s). A D^3 specifies the relationships between source and target objects including how much inconsistency can be tolerated between them before it is necessary to restore it, and the actions to take to maintain consistency when it reaches intolerable levels. All these D^3s together form an *Interdatabase Dependency Schema* (IDS) [3].

When a source object in a D^3 is updated, the consistency between itself and the target object in the same D^3 may be violated beyond allowable levels. Then, a system-generated transaction updates the target object to maintain consistency. The updated target object may participate as source in another D^3, therefore a series of related transactions may execute to maintain consistency. These related transactions form a transaction tree called a *polytransaction* [3,1,2].

A polytransaction tree contains nodes (corresponding to its component transactions), and edges (identifying the "execution mode" between the parent and children transactions) and is determined as

follows. For each D^3_ε IDS containing a source object updated by T, and in need to restore consistency with its target object, a new node is added corresponding to a new transaction T' (child of T). The operations of T' which restore consistency are the actions specified in the D^3.

Execution modes: A child transaction is *coupled* if the parent transaction must wait until the child transaction completes before proceeding further. It is *decoupled* otherwise. Also, a coupled transaction is *vital* (the parent transaction must fail if the child fails), or *nonvital* (the parent transaction survives the failure of a child).

Cross-references

- ▶ Database Dependencies
- ▶ Database Management System
- ▶ Distributed Transaction Management
- ▶ Extended Transaction Models
- ▶ Inconsistent Databases
- ▶ Transaction
- ▶ Transaction Management
- ▶ Transaction Manager

Recommended Reading

1. Karabatis G., Rusinkiewicz M., and Sheth A. Correctness and enforcement of multidatabase interdependencies, Advanced Database Systems, LNCS, vol. 759, Springer-Verlag, NY, 1993, pp. 337–358.
2. Karabatis G., Rusinkiewicz M., and Sheth A. Interdependent database systems. In Management of Heterogeneous and Autonomous Database Systems. A. Elmagarmid, M. Rusinkieuicz, and A. Sheth (eds.), Morgan-Kaufmann, San Francisco, CA, 1999, pp. 217–252.
3. Rusinkiewicz M., Sheth A., and Karabatis G. Specifying interdatabase dependencies in a multidatabase environment, IEEE Computer, 24(12): December 1991, pp. 46–53.

Port Binding

▶ Storage Security

Position Snapping

▶ Map Matching

Positive Infinity

▶ Forever

Positive Predictive Value

▶ Precision and Recall

Positive Relational Algebra

Cristina Sirangelo
University of Edinburgh, Edinburgh, UK

Synonyms

SPJRU-Algebra; SPCU-Algebra

Definition

Positive relational algebra is the fragment of relational algebra which excludes the difference operator. Relational algebra queries are expressions defining mappings from database instances of an input database schema to an output relation. In particular a positive relational algebra query over a database schema τ is one of the following expressions, each one with an associated set of attributes:

- A constant relation over a set of attributes U is a positive relational algebra query with associated set of attributes U;
- If $R(U)$ is a relation schema in τ, the relation symbol R is a positive relational algebra query with associated set of attributes U;
- If Q and Q' are positive relational algebra queries with sets of attributes U and U' respectively, the following are positive relational algebra queries:
  - The *selection* $\sigma_{A=B}(Q)$ or $\sigma_{A=c}(Q)$, with set of attributes U, where A and B are attributes in U, and c is a constant value;
  - The *projection* $\pi_X(Q)$, with set of attributes X, where $X \subseteq U$;
  - The *natural join* $Q \bowtie Q'$, with set of attributes $U \cup U'$;
  - The *renaming* $\rho_{U \to W}(Q)$, with set of attributes W;
  - The *union* $Q \cup Q'$, in the case that $U = U'$, with associated set of attributes U.

The semantics of a positive relational algebra query on a database instance I of schema τ is a relation instance defined as follows: the semantics of a constant relation is the relation itself; the semantics of a relation symbol R of τ is the value of R in I; the semantics of selection, projection, natural join, renaming and union expressions above, is defined according to the semantics of the corresponding operator on inputs given by the semantics of Q and Q'.

Key Points

The basic operators of the positive relational algebra are a non-redundant set: by removing any of these operators the set of expressible query mappings would be reduced. Moreover they allow the simulation of other operators. Among these, the intersection $R_1 \cap R_2$ of two relations over the same set of attributes can be simulated as $R_1 \bowtie R_2$; the generalized selection whose condition is a positive boolean combination of equality atoms can be simulated using composition and union of primitive selection operators; consequently also the theta-join, with the same restriction on the join condition, and the equijoin can be simulated.

An example of positive relational algebra query over a database schema consisting of relation schemas *Students*(*student-number, student-name*) and *Exams* (*course-number, student-number, grade*), is the expression $\pi_{student-name}(Students \bowtie \sigma_{grade = A}(Exams))$. It returns the names of the students that have passed at least one exam with grade A.

The positive relational algebra is equivalent (in that it expresses the same query mappings) to other query languages such as *unions of conjunctive queries* in safe relational calculus, and *nonrecursive datalog* with one target predicate.

In the absence of attribute names, the basic operators of positive relational algebra are $\{\sigma, \pi, \times, \cup\}$. The positive relational algebra without names turns out to be equivalent to the positive relational algebra with names and thus equivalent to the other above mentioned query paradigms.

Cross-references

- ▶ Difference
- ▶ Join
- ▶ Nonrecursive Datalog
- ▶ Projection
- ▶ Relation
- ▶ Relational Algebra
- ▶ Relational Calculus
- ▶ Renaming
- ▶ Selection
- ▶ Union

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases. Addison-Wesley, Reading, MA, 1995.

Possible Answers

Gösta Grahne
Concordia University, Montreal, QC, Canada

Synonyms

Maybe answer; Credulous reasoning; Consistent facts

Definition

In an incomplete database, which is a set of complete databases (possible worlds), the possible answer to a query is the set of tuples that are in the answer to the query, when posed on *some* possible world. The dual of the possible answer is the *certain* answer, which consists of all tuples true in the the answer to the query when posed simulatenously on *all* possible worlds.

Key Points

For more information on the certain and possible answers, as well as on various (partial) orders on incomplete databases, see [1,2].

Cross-references

- ▶ Certain (and Possible) Answers
- ▶ Incomplete Information
- ▶ Naive Tables

Recommended Reading

1. Grahne G. The Problem of Incomplete Information in Relational Databases. Springer, Berlin, 1991.
2. Libkin L. Aspects of Partial Information in Databases. PhD Thesis, University of Pennsylvania, 1994.

Postings File

- ▶ Inverted Files

Post-Randomization Method

► PRAM

PRAM

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Post-randomization method

Definition

The Post-RAndomization Method (PRAM) is a probabilistic, perturbative masking method for disclosure protection of categorical microdata. In the masked file, the scores on some categorical attributes for certain records in the original file are changed to a different score according to a prescribed probability mechanism, namely a Markov matrix. The Markov approach makes PRAM very general, because it encompasses noise addition, data suppression and data recoding.

Key Points

The PRAM matrix contains a row for each possible value of each attribute to be protected. This rules out using the method for continuous data. PRAM was invented by Gouweleeuw et al. [1]. The information loss and disclosure risk in data masked with PRAM largely depend on the choice of the Markov matrix and are still (open) research topics [2].

Cross-references

► Inference Control in Statistical Databases
► Microdata
► SDC Score

Recommended Reading

1. Gouweleeuw J.M., Kooiman P., Willenborg L.C.R.J., and DeWolf P.-P. Post randomisation for statistical disclosure control: Theory and implementation, 1997. Research Paper No. 9731 (Voorburg: Statistics Netherlands).
2. de Wolf P.-P. Risk, utility and PRAM. In J. Domingo-Ferrer and L. Franconi (eds.). Privacy in Statistical Databases LNCS, vol. 4302, 2006, pp. 189–204.

Precision

Ethan Zhang[1,2], Yi Zhang[2]
[1]University of California-Santa Cruz, Santa Cruz, CA, USA
[2]Yahoo! Inc., Santa Clara, CA, USA

Definition

Precision measures the accuracy of an information retrieval (IR) system. More precisely, precision is the fraction of retrieved documents that are relevant. Consider a test document collection and an information need Q. Let R be the set of documents in the collection that are relevant to Q. Assume an IR system processes the information need Q and retrieves a document set A. Let $|R_a|$ denote the number of documents that are in both R and A. Further let $|R|$ and $|A|$ be the numbers of documents in R and A, respectively. The *precision* of the IR system for Q is defined as $P = |R_a|/|A|$.

Key Points

Precision and recall are the most frequently used and basic retrieval performance measures. Many other standard performance metrics are based on the two concepts.

Cross-references

► Average Precision
► F-Measure
► 11-Point Precision-Recall Curve
► Precision-Oriented Effectiveness
► Recall
► Standard Effectiveness Measures

Precision and Recall

Ben Carterette
University of Massachusetts Amherst, Amherst, MA, USA

Synonyms

Positive predictive value; Sensitivity; False negative rate

Definition

Recall measures the ability of a search engine or retrieval system to locate relevant material in its index. *Precision* measures its ability to not rank nonrelevant

material. With everything above rank cut-off n considered "retrieved" and everything below considered "not retrieved," precision and recall can be stated mathematically as:

$$\text{precision} = \frac{|\text{retrieved} \,\&\, \text{relevant at rank } n|}{|\text{retrieved at rank } n|}$$

$$\text{recall} = \frac{|\text{retrieved} \,\&\, \text{relevant at rank } n|}{|\text{relevant}|}$$

Key Points

Precision and recall are the traditional metrics for retrieval system performance evaluation, and nearly all other performance measures can be seen as either precision-based, recall-based, or a combination of the two.

There is a trade-off between precision and recall: as more is done to bring more relevant results into a ranking, increasing recall, inevitably nonrelevavnt results will be captured as well, decreasing precision. This can be seen in precision-recall curves, which plot precision against recall while varying rank cut-off n.

		Condition	
		True	False
Prediction	True	True Positive	False Positive
	False	False Negative	True Negative

Precision and recall are not limited to information retrieval; they can be defined more formally on any 2×2 contingency table. For the table
precision is True Positives divided by True Positives + False Positives and recall is True Positives divided by True Positives + False Negatives. In medical literature, precision is known as *positive predictive value* and recall as *sensitivity*; in the statistics literature, recall is equivalent to the *power* (or 1 − Type II error rate) of a hypothesis test (precision has no analogue).

Cross-references

- ▶ Average Precision
- ▶ Precision at n
- ▶ Recall

Recommended Reading

1. van Rijsbergen C.J. Information Retrieval. Butterworths, London, UK, 1979.

Precision at n

Nick Craswell
Microsoft Research Cambridge, Cambridge, UK

Synonyms

P@n

Definition

In an information retrieval system that retrieves a ranked list, the top-n documents are the first n in the ranking. Precision at n is the proportion of the top-n documents that are relevant.

Key Points

If r relevant documents have been retrieved at rank n, then:

$$\text{Precision at n} = \frac{r}{n}$$

The value of n can be chosen based on an assumption about how many documents the user will view. In Web search a results page typically contains ten results, so n = 10 is a natural choice. However, not all users will use the scrollbar and look at the full top ten list. In a typical setup the user may only see the first five results before scrolling, suggesting Precision at 5 as a measure of the initial set seen by users. It is the document at rank 1 that gets most user attention, because this is the document that users view first, suggesting the use of Precision at 1 (which is equivalent to Success at 1).

It is possible to calculate precision at a later cutoff, although precision gives equal weight to every result in the list. For example, when calculating precision at 1,000 the 1,000th document is as important as the 1st, whereas users of a ranked retrieval system are likely to consider the 1st document most important. Other information retrieval measures place a greater emphasis on early ranks, such as Mean Average Precision and Mean Reciprocal Rank.

In set-based retrieval, where there is no ranking, the precision of a retrieved set of size n can still be calculated, as r/n.

In order to calculate precision at n it is only necessary to obtain relevance judgments for the top-n documents, unlike recall, which can only be measured if the complete set of relevant documents has been identified.

Cross-references

- ▶ Average Precision
- ▶ Precision-Oriented Effectiveness Measures
- ▶ Recall
- ▶ Success at n

Precision-Oriented Effectiveness Measures

Nick Craswell
Microsoft Research Cambridge, Cambridge, UK

Definition

Precision-oriented evaluation in information retrieval considers the relevance of the top *n* search results, for small *n* and using a set of relevance judgments that need not be complete. Such "shallow" evaluation is consistent with a user who only cares about the top-ranked documents. Relaxing the requirement of identifying all relevant documents for every query means that certain measures, such as recall at *n*, cannot be applied. However, it also allows evaluation on a very large corpus, where employing human relevance assessors to find the complete relevant set for each query would be too expensive. Both aspects of precision-oriented evaluation, the shallow viewing of results and the large corpus, are associated with Web search, where search results are typically a top-10 and the corpus may contain tens of billions of documents.

Historical Background

The Cranfield II experiments in 1963 were a landmark effort in information retrieval evaluation [3]. A test collection comprising 1,440 documents, 225 queries and relevance judgments was created. Because the document set was small, it was possible to judge the relevance of every document for each query. Using complete relevance judgments it is possible to evaluate precision, the proportion of retrieved documents that are relevant, and recall, the proportion of relevant documents that are retrieved. The test collection is reusable, in that it is possible to run a new retrieval experiment using existing queries and relevance judgments, without encountering any unjudged document in the search results.

Judging all documents for every query does not scale well with corpus size, and becomes untenable even with tens of thousands of documents. The best-known solution to this problem is the pooling method, as introduced in the Text Retrieval Conference (TREC) in 1991 [5]. Under pooling, the top-n documents are merged from a large number of systems. This pool of documents does not necessarily contain all relevant documents. However, if a sufficient variety of top-n lists are merged, for a large enough n, the pool will contain the majority of the relevant documents. After judging the pool, the set of relevance judgments can be thought of as "sufficiently complete" to be reusable. In other words, a new retrieval experiment may retrieve unjudged documents, but these can be assumed to be irrelevant since almost all the relevant documents have already been identified.

Most TREC experiments have resulted in test collections where the pools are sufficiently complete for reuse. However, for the largest TREC corpora, it is unlikely that the full relevant set has been identified, or could be identified due to practical limitations in the judging resources [4]. In such cases it is possible to judge with shallow pools, for example judging the top 20 documents and using a precision-oriented measure. Since the full relevant set has not been discovered, it is not straightforward to measure recall, or use any measure that relies on knowing the size of the relevant set. The test collection may not be reusable.

Foundations

To understand precision-oriented evaluation, it is useful to revisit the fundamental measures of information retrieval, precision and recall. If there are *R* known relevant documents, and *r* of them have been retrieved at rank n, then:

$$\text{Precision at n} = \frac{\text{r}}{\text{n}}$$

$$\text{Recall at n} = \frac{\text{r}}{\text{R}}$$

For a given cutoff n, these are just different ways of normalizing r. If r increases, because a new retrieval system retrieves a better set of *n* documents, both precision and recall improve.

One aspect of precision-oriented evaluation is where not all relevant documents have been identified, so R is unknown. In such a case it is not straightforward to estimate recall, which depends on knowledge of R. However, if the top n documents have been judged, it is possible to accurately measure precision.

Another aspect of precision-oriented evaluation is shallow evaluation. At an early cutoff (small n) few relevant documents will have been retrieved (small r),

but precision may still be quite high. By contrast, recall might be low, particularly if R is much higher than r and n. In a standard precision-recall curve, which shows precision and recall at multiple cutoffs, precision-oriented evaluation considers the left-hand end of the curve. Precision-oriented evaluation models a user who does not need to see all relevant documents and is impatient, unwilling to look deep into the ranking.

Some experiments have both shallow evaluation and incomplete judgments, for example the TREC Very Large Collection Track [4] judged with a pool depth of 20, to measure Precision at 20. Metrics such as Precision at n, Success at n, Mean Reciprocal Rank and Average Precision at n have been applied in such settings. More details about these metrics can be found from the corresponding entries in this encyclopedia.

In other experiments, judgments may be incomplete or evaluation shallow, but not both. In those cases, it is not clear that the evaluation should be called precision-oriented. It is possible to evaluate with a recall-oriented measure using incomplete judgments, via measures such as bpref and inferred AP [1,6]. It is also possible to perform shallow evaluation with complete judgments, particularly in cases with very few relevant documents such as known item search. In a setting where R tends to be between 1 and 10, a metric such as R-Precision could be thought of as precision-oriented.

Key Applications

Web search is probably the most well-known application where precision-oriented evaluation is used. Users do not often look deep into the results list, so evaluation that concentrates on shallow ranks is appropriate. In addition, the document collection is so large, that it is difficult to be sure that all good answers for a query have been identified. The exception is where a user has a very focused need, for example navigational search, where only a single document is required. Even then the web is volatile, with documents being created, modified and deleted without warning, so it is necessary to perform careful maintenance of a query set. For queries with multiple correct answers, it is usual to calculate precision-oriented measures, and very difficult to estimate recall.

Data Sets

Data sets for precision-oriented evaluation, and Information Retrieval experimentation in general, are available from the National Institute of Standards and Technology, in the Text Retrieval Conference (TREC) series [5]. TREC experiments typically involve hundreds of thousands of documents or more, and 50 or more query topics.

Cross-references

- ▶ Average Precision at n
- ▶ Bpref
- ▶ Mean Reciprocal Rank
- ▶ Precision
- ▶ Precision at n
- ▶ R-Precision
- ▶ Standard Effectiveness Measures
- ▶ Success at n

Recommended Reading

1. Buckley C. and Voorhees E.M. Retrieval evaluation with incomplete information. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 25–32.
2. Clarke C.L.A., Scholer F., and Soboroff I. The TREC 2005 terabyte track. In Proc. The 5th Text Retrieval Conference, 2005.
3. Cleverdon C. The Cranfield Tests on Index Language Devices. Readings in Information Retrieval. Morgan Kaufmann, San Fransisco, CA, 1997, pp. 47–59.
4. Hawking D. and Craswell N. The very large collection and web tracks. In TREC Experiment and Evaluation in Information Retrieval, E. Voorhees, D. Harman (eds.). MIT Press, Cambridge, MA, 2005, pp. 199–231.
5. Voorhees E.M. and Harman D.K. TREC: Experiment and Evaluation in Information Retrieval. MIT Press, Cambridge, MA, 2005.
6. Yilmaz E. and Aslam J.A. Estimating average precision with incomplete and imperfect judgments. In Proc. Int. Conf. on Information and Knowlege Management, 2006, pp. 102–111.

Predicate Calculus

- ▶ First-Order Logic: Semantics
- ▶ First-Order Logic: Syntax

Predicate Logic

- ▶ First-Order Logic: Semantics
- ▶ First-Order Logic: Syntax

Prediction of Event Occurrence

▶ Event Prediction

Prediction Regarding Future Events

▶ Event Prediction

Prefix Tree

▶ Trie

Presenting Structured Text Retrieval Results

Jaap Kamps
University of Amsterdam, Amsterdam,
The Netherlands

Synonyms

Structured Text Retrieval Tasks

Definition

Presenting structured text retrieval results refers to the fact that, in structured text retrieval, results are not independent and a judgment on their relevance needs to take their presentation into account. For example, HTML/XML/SGML documents contain a range of nested sub-trees that are fully contained in their ancestor elements. As a result, structured text retrieval should make explicit the assumptions on how the retrieval results are to be presented. Four of the main assumptions to be addressed are the following. First, the *unit of retrieval* assumption: is there a designated retrieval unit (such as the document or root node of the structured document) or can every sub-tree be retrieved in principle? Second, the *overlap* assumption: may retrieval results contain text or content already part of other retrieval results (such as a full article and one of its individual paragraphs)? Third, the *context* assumption: can results from the same structured document be interleaved with results from other structured documents? Fourth, the *display* assumption: is a retrieval result (say a document sub-tree corresponding to a paragraph) presented as an autonomous unit of text, or as an entry-point within a structured document?

Historical Background

Although similar considerations play an important role in the design of user interfaces (see, for example, [6]), this entry will focus on the underlying principles of the different structured text retrieval tasks. Structured text retrieval dates back, at least, to the early days of passage retrieval [14]. Early passage retrieval approaches have been using either the document structure (sentences, paragraphs, sections, etc.), or arbitrary text windows of fixed length. The early experimental results primarily confirmed the effectiveness of passage-level evidence for boosting document retrieval. The use of document structure derived from SGML mark-up was pioneered by [20], studying adhoc SGML element retrieval. Probabilistic indexing approaches for databases have been studied even earlier [4], allowing to rank results based on vague queries. Adhoc XML element retrieval and best entry point retrieval was studied in the Focus project [3,10].

The main thrust in recent years is the initiative for the evaluation of XML retrieval [7]. The retrieval task descriptions heavily evolved during the different years. Initially, in 2002, INEX studied adhoc XML element retrieval for keyword (Content-Only) and structured (Content-And-Structure) queries with the goal to "retrieve the most specific relevant document components" [5, p. 2]. This generic adhoc XML element retrieval task was continued at INEX 2003 [9, p. 200] and at INEX 2004 [12, p. 237], asking for "components that are most specific and most exhaustive with respect to the topic of request." Ongoing discussion, and vivid disagreement, on the interpretation of generic adhoc XML element retrieval task prompted the introduction of three different retrieval strategies at INEX 2005 [11, pp. 385–386]: *Thorough* aims to find all highly exhaustive and specific elements (roughly corresponding to the earlier INEX task); *Focussed* aims to find the most specific and exhaustive element in path (no overlapping results); and *Fetch and browse* aims to first identify relevant articles, and then to find the most specific and exhaustive elements within the fetched articles (results grouped by article). These different adhoc XML element retrieval tasks have been

continued and further explicated at INEX 2006 [1], with the Fetch and browse task refined to: *Relevant in Context* aims to retrieve a set of non-overlapping relevant elements per article; and *Best in Context* aims to retrieve, per article, a single best entry point to read its relevant content. At INEX 2007 three tasks are continued: Focused, Relevant in Context, and Best in Context, but liberalized to arbitrary passages [2].

Foundations

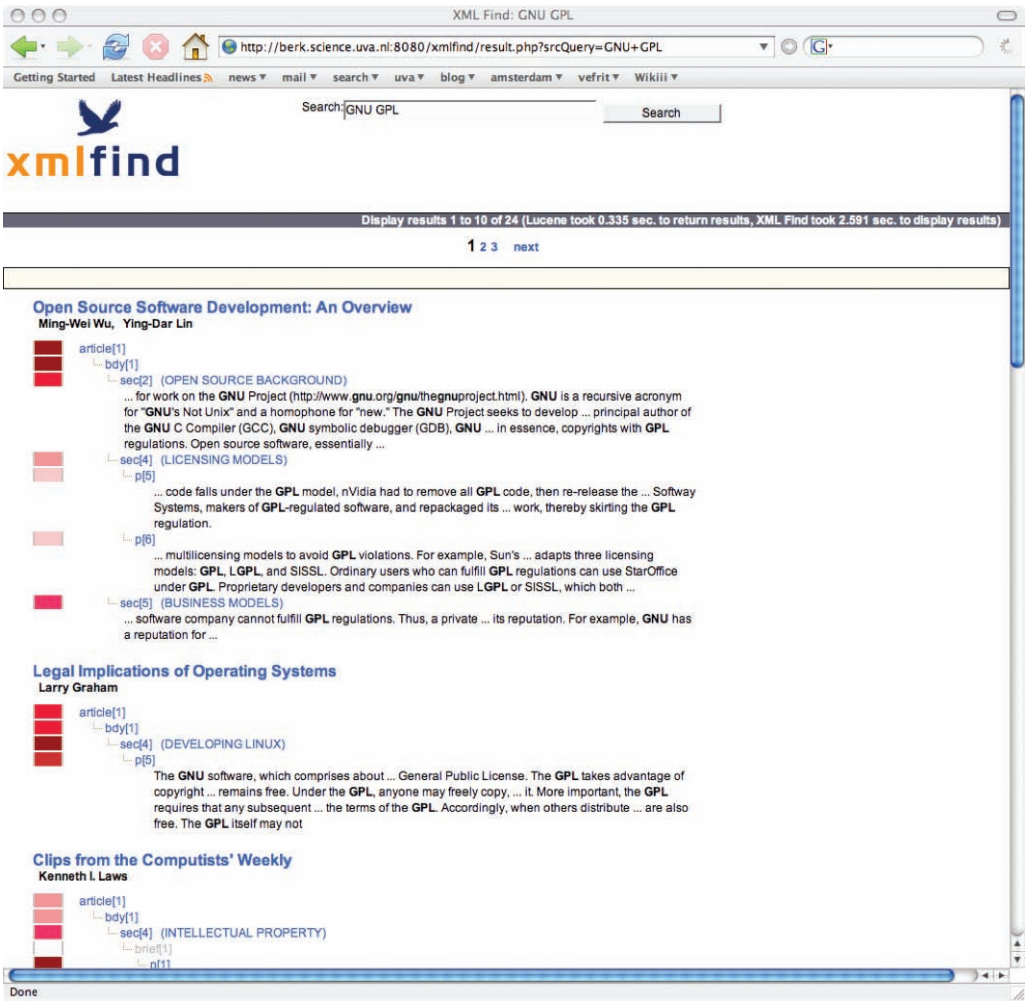
The way in which retrieval results are presented to users, is always a crucial factor determining the success or failure of an operational retrieval system. However, within the Cranfield/TREC tradition of evaluating document retrieval systems it is unproblematic to abstract away from presentation issues and analyze retrieval effectiveness by regarding retrieved documents as atomic and independent results. In structured text retrieval, the situation is different, and there is a need to make explicit some of the assumptions underlying the retrieval task since these have an impact on what is regarded as a "relevant" retrieval result.

First, the *unit of retrieval* assumption: is there a designated retrieval unit (such as the document or root node of the structured document) or can every sub-tree be retrieved in principle? Rather than treating documents as atomic, structured documents have internal document structure that allows any logical unit of them to be retrieved. For example, in case of a textual document where the layout structure is marked up, it is possible to retrieve sections, paragraphs, or still the whole article if its completely devoted to the topic of request. Figure 1 contains a screen-shot of a XML element retrieval system that retrieves a ranked list of XML elements.

Second, the *overlap* assumption: may retrieval results contain text or content already part of other retrieval results (such as a full article and one of its individual paragraphs)? Interactive experiments at INEX 2004 [17] clearly revealed that test persons disliked a ranked list of element results that overlap in whole or part in their content. Hence, if the retrieval tasks should reflect a scenario in which the ranked elements are directly displayed to an end-user, retrieval results should be disjoint.

Third, the *context* assumption: can results from the same structured document be interleaved with results from other structured documents? A further finding of the interactive experiments at INEX 2004 [17] is that test persons prefer results from the same document be grouped together. Figure 2 contains a screen-shot of a XML element retrieval system that retrieves XML elements displayed in document order in their article's context.

Presenting Structured Text Retrieval Results. Figure 1. Displaying structured text retrieval results as a ranked list of elements (reproduced from [13]).



Presenting Structured Text Retrieval Results. Figure 2. Displaying structured text retrieval results within article context (reproduced from [15]).

Fourth, the *display* assumption: is a retrieval result (say a document sub-tree corresponding to a paragraph) presented as an autonomous unit of text, or as an entry-point within a structured document? A decision on the relevance of a particular document component crucially depends on whether it will be presented as an isolated excerpt, or within its original context. In the first case, the component should be fully self-contained: it should not only contain the relevant information (say, for example, a description of an algorithm) but also establish that this information is, indeed, satisfying the topic of request (for example, that the algorithm is the fastest way to lexicographically sort a list, if that were the topic of request). This is related to linguistics, where there is a common distinction between the context (or topic/theme: that what is being talked about), and the information (or comment/rheme/focus: that what is being said). If results are to be presented in their document context, the link to the topic of request can be taken for granted and only the sought information can be regarded as relevant. If results are to be presented out of context, both the information and its relation to the topic of request are needed to establish the relevance of a document component.

Table 1 shows how the different structured text retrieval tasks are based on different underlying assumptions. For traditional document or article retrieval, there is a fixed unit of retrieval and assumptions on overlap, context, or display do not apply. For the generic adhoc element retrieval task (INEX 2002–2004) or Thorough (INEX 2005–2006), any document

Presenting Structured Text Retrieval Results. Table 1. Structured text retrieval tasks

	Unit of retrieval	Overlap	Context	Display
Article retrieval	Whole article	–	–	–
Thorough	Arbitrary element	Allowed	Scattered	Elements
Focussed/Focused	Arbitrary element	Non-overlapping	Scattered	Elements/Passages
Fetch and browse	Arbitrary element	Allowed	List per article	Elements
Relevant in context	Arbitrary element	Non-overlapping	Set per article	Elements/Passages
Best in context	Arbitrary element	Non-overlapping	One result per article	Starting point

component can be retrieved, and there are no restrictions on overlap, context, or display. Basically, the task is system-biased, reflecting the ability of the retrieval engine to estimate the relevance of individual document components, for example for further processing methods. For Focussed/Focused (INEX 2005–2007), a ranked list of non-overlapping document components is asked for, with no restrictions on context or display. This task reflects a scenario where a ranked-list of document components is directly presented to the searcher. For Fetch and browse (INEX 2005), retrieval results from the same structured document need to be returned consecutive, with no restriction on overlap or display. This results in a tasks resembling on the one hand traditional document retrieval, whilst on the other hand providing deep-linking to relevant document components. The same holds for Relevant in context (INEX 2006–2007), where there is an unranked set of now non-overlapping elements per article, reflecting results to be presented in document order. Finally, Best in context (INEX 2006–2007) explicitly asks for a single best entry point into the article (so non-overlapping and non-scattered articles by definition). This scenario captures a "relative" notion of relevance, where users desire access to the best information, rather than all relevant information.

These different retrieval tasks lead to different evaluations of what systems and techniques are effective for structured text retrieval. Although these tasks are not unrelated, for example, the generic Thorough task (capturing the ability of a system to estimate the relevance of an element) can be use as input for further processing for the other tasks, each of these different retrieval tasks is capturing a different aspect of structured text. The retrieval tasks bring in elements from the task context in which they are to be applied, either in a end-user setting or system setting. As a result, the richer descriptions of the task's context and underlying assumptions are resonating more closely with actual real-world applications [19]. Bringing task-specific elements into information retrieval benchmark testing has been identified as one of the main research directions for further enhancing information access in general [18].

Key Applications

Structured text retrieval has the potential to improve information access by giving more direct access to the relevant information inside documents. As [14, p. 49] put it:

> ▶ Large collections of full-text documents are now commonly used in automated information retrieval. When the stored document texts are long, the retrieval of complete documents may not be in the users' best interest. In such circumstances, efficient and effective retrieval results may be obtained by using passage retrieval strategies designed to retrieve text excerpts of varying size in response to statements of user interest.

Structured document retrieval is becoming increasingly important in all areas of information retrieval, the application to full-text book searching is obvious and such commercial systems already exist [19].

Future Directions

Improving information access by formulating retrieval tasks that capture interesting aspects of real-world structured text searching is an ongoing open problem. There has been a series of workshops addressing open problems, including real-world applications, the unit of retrieval, tasks and measures, and the problem of overlap [18].

The traditional picture of IR takes as input a document collection and a query, and gives as output a ranked list of documents. In the retrieval task, there

no distinction between the hit list (communicating the ranked list) and the actual result documents. Where structured document retrieval is going beyond a linear ranked list of results, at least conceptually, interesting new research questions present themselves. By presenting related results from the same article, like in Fig. 2, the hit-list becomes a query-biased summary of the discourse structure of the retrieved article. [16] conduct experiments on the level of detail desired by searchers. Evaluation of such a system seems to require taking both retrieval effectiveness and document summarization aspects into account.

Data Sets

Notable data-sets are:

1. The *Shakespeare test collection* used in the Focus project 2000–2001 [3].
2. The *IEEE Computer Society collection* used at INEX 2002–2004 [7].
3. The expanded *IEEE Computer Society collection* used at INEX 2005 [7].
4. The *Wikipedia XML Corpus* used at INEX 2006–2007 [7].

Cross-references

- ► Evaluation Metrics for Structured Text Retrieval
- ► INitiative for the Evaluation of XML Retrieval
- ► XML Retrieval

Recommended Reading

1. Clarke C., Kamps J., and Lalmas M. INEX 2006 retrieval task and result submission specification. In INEX 2006 Workshop Pre-Proceedings, 2006, pp. 381–388.
2. Clarke C.L.A., Kamps J., and Lalmas M. INEX 2007 retrieval task and result submission format. In Pre-Proceedings of INEX, 2007, pp. 445–453.
3. Focus. Focussed Retrieval of Structured Documents – A Large Experimental Study, 2001.
4. Fuhr N. A probabilistic framework for vague queries and imprecise information in databases. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 696–707.
5. Govert N. and Kazai G. Overview of the INitiative for the Evaluation of XML retrieval (INEX) 2002. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, pp. 1–17.
6. Hearst M.A. User interfaces and visualization, Chapter X. In Modern Information Retrieval. ACM, New York, NY, USA, 1999, pp. 257–324.
7. INEX. INitiative for the Evaluation of XML Retrieval, 2007. http://inex.is.informatik.uni-duisburg.de/.
8. Jones K.S. What's the value of TREC – is there a gap to jump or a chasm to bridge? SIGIR Forum, 40(1):10–20, 2006.
9. Kazai G., Lalmas M., Govert N., and Malik S. INEX'03 retrieval task and result submission specification. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, pp. 200–203.
10. Kazai G., Lalmas M., and Reid J. Construction of a test collection for the focussed retrieval of structured documents. In Proc. 25th European Conf. on IR Research, 2003, pp. 88–103.
11. Lalmas M. and Kazai G. INEX 2005 retrieval task and result submission specification. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 385–390.
12. Lalmas M. and Malik S. INEX 2004 retrieval task and result submission specification. In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2004, pp. 237–240.
13. Malik S., Klas C.-P., Fuhr N., Larsen B., and Tombros A. Designing a user interface for interactive retrieval of structured documents – lessons learned from the INEX interactive track. In Proc. 10th European Conf. Research and Advanced Technology for Digital Libraries, 2006, pp. 291–302.
14. Salton G., Allan J., and Buckley C. Approaches to passage retrieval in full text information systems. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 49–58.
15. Sigurbjörnsson B. Focused Information Access using XML Element Retrieval. SIKS dissertation series 2006–2028, University of Amsterdam, 2006.
16. Szlávik Z., Tombros A., and Lalmas M. Feature- and query-based table of contents generation for XML documents. In Proc. 29th European Conf. on IR Research, 2007, pp. 456–467.
17. Tombros A., Larsen B., and Malik S. The interactive track at INEX 2004. In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrival, 2004, pp. 410–423.
18. Trotman A., Geva S., and Kamps J. (eds.). In Proc. of the SIGIR 2007 Workshop on Focused Retrieval, 2007.
19. Trotman A., Pharo N., and Lehtonen M. XML-IR users and use cases. In Proc. 5th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 400–412.
20. 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.

Preservation

- ► Archiving Experimental Data

Preserving Database Consistency

- ► Correctness Criteria Beyond Serializability

Preview

► Result Display

Primary Index

Yannis Manolopoulos[1], Yannis Theodoridis[2], Vassilis J. Tsotras[3]
[1]Aristotle University of Thessaloniki, Thessaloniki, Greece
[2]University of Piraeus, Piraeus, Greece
[3]University of California-Reverside, Riverside, CA, USA

Synonyms

Clustering index; Sparse index

Definition

A tree-based index is called a *primary index* if its order is the same as the order of the file which it indexes Consider a relation *R* with some numeric attribute *A* taking values over an (ordered) domain *D*. Assume that relation *R* has been physically stored as an *ordered* file, following the order of the values of attribute *A*. Furthermore, assume that a tree-based index (e.g., B +-tree) has been created on attribute *A*. Then this index is primary.

Key Points

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

Note that a primary index also controls the physical placement of the data records in a file. Since such an index also clusters the values of the file, the term *clustering* index has alternatively been used. In contrast, the order in the leaf pages of a secondary index is not the same as the order of the records in the file. Hence, a secondary index (also called *non-clustering*) needs an extra level of indirection, namely, a pointer to the actual position of a record with a given value in the relation file. In other words, a secondary index only clusters references to records (in the form of <value, pointer> fields), but *not* the records themselves. This extra indirection from a leaf page of a secondary index to the actual position of a record in a file has important subsequences on optimization. Consider for example a secondary index (B + -tree) on the *ssn* attribute of the *Employee* relation (which assume is ordered by the *name* attribute). A query that asks for the salaries of employees with *ssn* in the range *[x,y]* can facilitate the B + -tree on *ssn* to retrieve references to all records in the query range. Assume there are 1,000 such *ssn* values in the *Employee* file. Since the actual *Employee* records must be retrieved (so as to report their salaries), each such reference needs to be materialized by possibly a separate page I/O (since the actual records can be in different pages of the *Employee* file). If instead the file was ordered on the *ssn* attribute, the B + -tree on *ssn* would have clustered (as primary index) the *Employee* records on *ssn*, and thus the 1,000 records that need to be retrieved would be located within few pages.

In practice, the search-key of a primary index is usually the file's primary key, however this is not necessary. That is, primary indexes need not be on the primary keys of relations. (In the above example, it was first assumed that the *Employee* file is ordered according to the *name* attribute and not according to the primary key *ssn* attribute). A relation can have several indices, on different search-keys; among them, at most one is primary (clustering) index and the rest are secondary ones.

Cross-references

► Access Path
► Index Creation and File Structures
► Index Sequential Access Method (ISAM)
► Secondary Index

Recommended Reading

1. Elmasri R. and Navathe S.B. Fundamentals of Database Systems, 5th edn. Addisson-Wesley, Boston, MA, 2007.

2. Manolopoulos Y., Theodoridis Y., and Tsotras V.J. Advanced Database Indexing. Kluwer, Dordrecht, 1999.
3. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edn. McGraw-Hill, New York, 2003.

Primary Memory

► Main Memory

Primitive Event

► Atomic Event

Principal Component Analysis

Heng Tao Shen
The University of Queensland, Brisbane, QLD, Australia

Synonyms

PCA

Definition

Principal components analysis (PCA) is a linear technique used to reduce a high-dimensional dataset to a lower dimensional representations for analysis and indexing. For a dataset P in D-dimensional space with its principal component set Φ, given a point $p \in P$, its projection on the lower d-dimensional subspace can be defined as: p. Φ_d, where Φ_d represents the matrix containing 1st to d^{th} largest principal components in Φ and $d < D$.

Key Points

PCA finds a low-dimensional embedding of the data points that best preserves their variance as measured in the high-dimensional input space [1]. It identifies the directions that best preserve the associated variances of the data points while minimize "least-squares" (Euclidean) error measured by analyzing data covariance matrix. The first principal component is the eigenvector corresponding to the largest eigenvalue of the dataset's co-variance matrix, the second component corresponds to the eigenvector with the second largest eigenvalue and so on. The chosen principal components form the lower dimensional subspace of interest. Typically, the top *d* principal components are selected since they carry the most information of original data. The lower dimensional representation for a point is then generated by multiplying the data point with the selected principal components. PCA makes a stringent assumption of orthogonality to make it amenable to linearity.

PCA has been widely used in many applications for exploratory data analysis and compression, making predictive models, indexing, information retrieval, etc.

Cross-references

► Dimensionality Reduction
► Discrete Fourier Transform
► Discrete Wavelet Transform and Wavelet Synopses
► Independent Component Analysis
► Isometric Feature Mapping
► Latent Semantic Indexing
► Locality-Preserving Mapping
► Locally Linear Embedding Laplacian Eigenmaps
► Multidimensional Scaling
► Semantic Subspace Projection
► Singular Value Decomposition

Recommended Reading

1. Jolliffe I.T. Principlal Componet Analysis. 2nd edn. Springer, New-York, 2002.

Principle of Locality

► Memory Locality

Privacy

Patrick C. K. Hung[1], Vivying S. Y. Cheng[2]
[1]University of Ontario Institute of Technology, Oshawa, ON, Canada
[2]Hong Kong University of Science and Technology, Hong Kong, China

Definition

Privacy helps to establish personal autonomy and create individualism. Privacy is a state or condition of

limited access to a person (e.g., client). In particular, information privacy relates to an individual's right to determine how, when, and to what extent information about the self will be released to another person or to an organization.

Key Points

With the rising occurrence of information privacy violations, people have begun to take interest in how, when, and where their personal information is being used. People usually interchange the concepts of security with privacy. In fact, they are essentially two different concepts used to protect data. In general, security involves the use of cryptographic tools to protect information in terms of confidentiality, availability, and access control and integrity enforcement. Privacy mainly focuses on the ability of keeping data away from public access, and the way to protect it according to the individual rights.

There are various definitions of privacy in the literature. Some researchers describe confidentiality as privacy, while some regard privacy as an aspect different from using cryptographic tools. Anderson [1] defined privacy as secrecy for benefit of the individual, and confidentiality as secrecy for the benefit of the organization. Alternatively, privacy can also be described by the ability to have control over the collection, storage, access, communication, manipulation and disposition of data. Privacy is also referred as the right for individuals to determine for themselves when, how, and to what extent information about them is communicated to others. As Westin [2] notes,

> *No definition [of privacy]... is possible, because [those] issues are fundamentally matters of values, interests and power.*

It can be said that privacy is a much broader concept than security; privacy protection is based on security protection. Security may enable privacy protection from authorized access, but security alone cannot provide privacy.

To enhance the privacy protection of personal information, legislative schemas and practice guidelines have been proposed by different organizations such as the Health Insurance Portability and Accountability Act (HIPAA) in the United States of America (USA) and the European Union (EU) Data Protection Act. These legislations and requirements can be abstracted as access control policies or rules to serve as a privacy foundation to control access to personal information.

Cross-references

- ▶ Data Privacy and Patient Consent
- ▶ Privacy-enhancing technologies
- ▶ Privacy Metrics
- ▶ Privacy policies and preferences
- ▶ Privacy-preserving data mining

Recommended Reading

1. Anderson R.J. A security policy model for clinical information systems. In Proc. 1996 IEEE Symp. on Security and Privacy, 1996.
2. Westin A. Privacy and freedom. New York, Atheneum, 1967.

Privacy Measures

▶ Privacy Metrics

Privacy Metrics

Chris Clifton
Purdue University, West Lafayette, IN, USA

Synonyms

Privacy measures

Definition

Measures to determine the susceptibility of data or a dataset to revealing private information. Measures include ability to link private data to an individual, the level of detail or correctness of sensitive information, background information needed to determine private information, etc.

Historical Background

Legal definitions of privacy are generally based on the concept of *Individually Identifiable Data*. Unfortunately, this concept does not have a clear meaning in the context of many database privacy technologies. The official statistics (census) community has long been concerned with measures for privacy, particularly in the contexts of microdata sets (datasets that represent

real data, but obscured in ways to protect privacy) and tabular datasets. Measures have largely been based on the probability that a specific value belongs to a given individual, given the disclosed data. As technologies have been developed to anonymize and analyze private data, measures have been developed to quantify the potential for disclosure of private information by those techniques. In 2001, Samarati and Sweeney published papers on *k*-anonymity as a measure of individual identifiability, leading to considerable research in anonymity measures. At the same time, the rise in privacy-preserving data mining techniques lead to measures based on the ability to estimate values from disclosed data. As of this writing, this field is still open, with many competing measures, often tied to specific anonymization, analysis, or privacy protection techniques.

Foundations

There have been two distinct approaches to measuring privacy, or more specifically probability of disclosure. The first is measuring anonymity. *k*-anonymity states that any released data item must be linked to at least *k* individuals with equal probability. It builds on a concept of *Quasi-Identifier QI*: a quasi-identifier is information that an adversary can use to link a released data record to an individual. Formally,

k-Anonymity [8] A given table T^* is said to satisfy *k*-anonymity if and only if each sequence of values in $T^\star[QI_T{}^\star]$ appears at least *k* times in $T^\star$.

While simple to compute, *k*-anonymity does not by itself guarantee that sensitive data is protected. As first pointed out in [7], if all *k* values with the same quasi-identifiers also have the same value for a sensitive attribute, *k*-anonymity can be met while still revealing the sensitive value for an attribute to an adversary. *k*-anonymity can be extended to deal with this issue:

ℓ-Diversity Principle [4] A $q^\star$-block is ℓ-diverse if contains at least ℓ "well-represented" values for the sensitive attribute *S*. A table is ℓ-diverse if every $q^\star$-block is ℓ-diverse.

This still does not resolve the issue, as real values may have skewed distributions; a particular rare value may be "well-represented" by occurring in only one $q^\star$-block, identifying the individuals represented in that block as having an unusually high probability of possessing that rare value. A further refinement is:

The t-closeness Principle [3] An equivalence class is said to have *t*-closeness if the distance between the distribution of a sensitive attribute in this class and the distribution of the attribute in the whole table is no more than a threshold *t*. A table is said to have *t*-closeness if all equivalence classes have t-closeness.

A challenge with any of these metrics is choosing appropriate values for *k*. Achieving a suitably low risk of disclosure may require a high value of *k* to protect particularly easy to identify individuals, resulting little specificity (and thus low value) in the anonymized dataset. This leads to a second set of metrics for anonymity, measuring not privacy but the fidelity of the disclosed data. These are typically based on the levels of generalization required to achieve a given level of anonymization, but as yet it isn't clear how well they relate to actual data utility (see [6]).

An alternative is to measure specifically the risk of identifying individuals, as with

δ-Presence[6] Given an external public table *P*, and a private table *T*, *δ-presence* holds for a generalization $T^\star$ of *T*, with $\delta = (\delta_{min}, \delta_{max})$ if

$$\delta_{min} \leq \mathcal{P}(t \in T \mid T^*) \leq \delta_{max} \qquad \forall\, t \in P.$$

The above approaches measure the ability of an adversary to link a data item to a specific individual. An alternative is to measure the ability to estimate sensitive data for an individual, once such linkage is performed. In [2], the assumption was that individuals provided sensitive data directly to the data analyst; the link between data and individual was thus known. Instead, the sensitive data is distorted to prevent the analyst from knowing values for an individual. This leads to a metric based on bounding the knowledge gained by an adversary from seeing the (distorted) sensitive value. The metric in [2] was based on two things: an interval and confidence level. If the analyst can determine that a value lies within the range $[x_1, x_2]$ with *c*% confidence, then the privacy at that confidence level is $x_2 - x_1$.

This measure raises two issues. First (as with ℓ-diversity), it relies on two numbers. This increases the difficulty of using it in practice. Second, as pointed out in [1], knowledge of the distribution of the original data (acquired by the analyst from the distorted dataset) may allow tightening of the bounds. They propose a measure based on the *differential entropy* of a random variable. The differential entropy $h(A)$ is a measure of the uncertainty inherent in *A*. Their metric for privacy is $2^{h(A)}$. Specifically, when adding noise from a random variable *A*, the privacy is:

$$\Pi(A) = 2^{-\int_{\Omega_A} f_A(a) log_2 f_A(a) da}$$

where Ω_A is the domain of A. This metric has several nice features. It is intuitively satisfying for simple cases. For noise generated from A, a uniformly distributed random variable between 0 and a, $\Pi(A) = a$. Thus this privacy metric is exactly the width of the unknown region. Furthermore, if a sequence of random variables A_i converges to B, then $\Pi(A_i)$ converges to $\Pi(B)$. For most random variables, e.g., a gaussian, the notion of width of the unknown region does not make sense. However, by calculating Π for such random variables, the above properties can be used to make the case that the privacy is equivalent to having no knowledge of the value except that it is within a region of width Π. This gives an intuitively satisfying way of comparing the privacy of different methods of adding random noise.

The authors extend this definition to *conditional privacy*, capturing the possibility that the inherent privacy from obscuring data may be reduced by what can be learned from a collection. The conditional privacy $\Pi(A|B)$ follows from the definition of conditional entropy:

$$\Pi(A|B) = 2^{-\int_{\Omega_{A,B}} f_{A,B}(a,b) log_2 f_{A|B=b}(a) da db}$$

They show how this can be applied to measure the actual privacy after reconstructing distributions of the original data to improve the accuracy of decision trees build on the obscured data [1,2].

Another use of this metric is to evaluate the inherent loss of privacy caused by data mining results. The use of conditional privacy enables estimating how much privacy is lost by knowing the data mining results, even with a "perfect" privacy-preserving technique such as secure multiparty computation. The literature has not yet addressed this issue; the assumption has generally been that the data mining results do not of themselves violate privacy.

Numerous other metrics have been proposed; the above give a representative view of the approaches and challenges faced in measuring privacy.

Key Applications

Primary applications are in aggregate analysis of privacy-sensitive data, such as individual health-care, personal preference, or financial information. Direct access to individual data is typically a confidentiality and access control issue; access to data is either allowed or it is not. Privacy measures come into play when individual data is analyzed as part of a broader study, where the end result is not solely for the benefit of the individuals whose data was used. (e.g., Studies of demographics or medical research.) Privacy laws typically control the use of data when not specifically to complete a transaction on behalf of the individual; measures are thus needed to show that privacy is maintained.

Future Directions

One approach is measures that correlate with legal and regulatory standards (e.g., the HIPAA safe harbor rules). Perhaps more promising as a research direction is risk-based measures such as δ-presence [5]. Location data (as from mobile devices) also poses new issues; European Community rules specifically discuss protection of location, but it is not clear to what extent existing measures can be applied. Another need is measures that adjust for differences in individual privacy needs, e.g., protecting outliers.

Cross-references

- ▶ Individually Identifiable Data
- ▶ Privacy
- ▶ Privacy-Preserving Data Mining
- ▶ Randomization Methods to Ensure Data Privacy
- ▶ Statistical Disclosure Limitation for Data Access

Recommended Reading

1. Agrawal D. and Aggarwal C.C. On the design and quantification of privacy preserving data mining algorithms. In Proc. 20th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2001, pp. 247–255.
2. Agrawal R. and Srikant R. Privacy-preserving data mining. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 439–450.
3. Li N. and Li T. T-closeness: privacy beyond k-anonymity and l-diversity. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
4. Machanavajjhala A., Gehrke J., Kifer D., and Venkitasubramaniam M. *l*-diversity: privacy beyond *k*-anonymity. ACM Trans. Knowl. Discov. Data, 1(1), No.3, March 2007.
5. Nergiz M., Atzori M., and Clifton C. Hiding the presence of individuals from shared databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 665–676.
6. Nergiz M.E. and Clifton C. Thoughts on k-anonymization. Data Knowl. Eng., 63(3):622–645, December 2007.
7. Øhrn A. and Ohno-Machado L. Using boolean reasoning to anonymize databases. Artif. Intell. Med., 15(3):235–254, 1999.
8. Sweeney L. Achieving *k*-anonymity privacy protection using generalization and suppression. Int. J. Uncertainty Fuzziness Knowl.-based Syst., 10(5):557–570, 2002.

Privacy Policies and Preferences

Patrick C. K. Hung, Yi Zheng, Stephanie Chow
University of Ontario Institute of Technology, Oshawa, ON, Canada

Definition

Privacy policies describe an enterprise's data practices, what information is collected from individuals (subjects), what the information (objects) will be used for, whether the enterprise provides access to the information, who the recipients are of any result generated from the information, how long the information will be retained, and who will be informed in the event of a dispute. A subject releases his or her data to the custody of an enterprise while consenting to the set of purposes for which the data may be used. The subject can express his or her preferences in a set of preference-rules to make decisions regarding the acceptability of privacy policies.

Historical Background

People have been concerned with privacy policies and preferences for more than 200 years. For example, the Hippocratic Oath was written as a guideline of medical ethics for doctors in respect to a patient's health condition, and states as follows: "*Whatsoever things I see or hear concerning the life of men, in my attendance on the sick or even apart there from, which ought not to be noised abroad, I will keep silence thereon, counting such things to be sacred secrets.*" Privacy policies and preferences are often expressed in natural language to specify or regulate how a system or an organization is to preserve privacy. In response to privacy policy and preference violations, many countries have enacted legislation to protect privacy for individuals. There are also different legislations for different industry sectors. For example, in the financial sector the Gramm-Leach-Bliley Act (GLB Act) of the U.S. requires banks to implement security programs to protect customer information; in the healthcare sector the Health Insurance Portability and Accountability Act (HIPAA) of the U.S. sets an American national standard to protect and enhance the rights of consumers, clients, and patients to control how their health information is used and disclosed. Similarly, the Personal Information Protection and Electronic Documents Act (PIPEDA) of Canada sets out ground rules for how private sector organizations may collect, use, or disclose personal information in the course of commercial activities. Failing to comply with these legislations, in their respective countries, may lead to civil and/or criminal penalties and/or imprisonment. In addition to the penalties, organizations may even suffer the loss of reputation and goodwill when the non-compliance of legislation is publicized. Looking at the above factors, it is evident that the privacy legislations have forced a widespread impact on the privacy policies and preferences.

Foundations

One can imagine that information privacy is usually concerned with the confidentiality of information. Threats to information privacy can come from insiders and from the outsiders in each enterprise. Privacy control is usually not concerned with individual subjects. Privacy policies are often expressed in natural language to specify or regulate how a system or an enterprise is to preserve privacy. Here are the principles of information privacy protection:

- Principle 1: Data-level security protection principle
  - Principle 1.1: Personal information shall be protected by security safeguards in a secure way such that data confidentiality, integrity, and availability can be achieved.
  - Principle 1.2: In needs to be accurate, complete, and up-to-date. The correction of inaccurate information should be allowed for maintaining data quality.
- Principle 2: Communication-level security protection principle
  - Principle 2.1: Information shall be transported in a secure way that data confidentiality and authentication of users/systems/services are achieved.
  - Principle 2.2: Data integrity must be maintained during the transmission in the communication channel.
- Principle 3: Consent requirement principle
  - Principle 3.1: Consent must be given by the owner of the information for the collection, disclosure, use, and retention of his/her information.
  - Principle 3.2: Owners of information should be allowed to review, control, and setup restrictions to their information on collection, usage, disclosure, and retention.

Principle 3 includes four limitations to specify how personal information can be released upon request:

- *Limitation on Collection*: The collection of personal information shall be limited to specific legitimate purposes of collection only.
- *Limitation on Disclosure*: The owner of information should be able to make special restrictions on the disclosure of his/her own information.
- *Limitation on Use*: The use of personal information shall be identified as legitimate use by the services provider and/or the owner of information.
- *Limitation on Retention*: Personal information shall be retained for only as long as the purpose for which it is used.

A subject releases his or her data to the custody of an enterprise while consenting to the set of purposes for which the data may be used. The subject can express his or her preferences in a set of preference-rules to make decisions regarding the acceptability of privacy policies. Preference assumes a real or imagined choice between alternatives and the possibility of rank ordering of these alternatives in a privacy policy. Privacy preferences are formally expressed by a set of rules and should preferably be captured through an interface. Subjects who are aware of how their personally identifiable information is being used, collected or disclosed can better understand how to protect it. Personally Identifiable Information (PII) includes information that identifies and individual and information that an organization may be able to identify. This includes, but is not limited to a subject's name, address, telephone number, social insurance number and credit card number(s). Among the most sensitive PII are medical and financial records.

Key Applications

In many instances, subjects do not have the knowledge or understanding of how his or her PII is being used, collected or disclosed, nor what privacy preferences are available. The adoption of information technology and the Internet have further added to this complexity. Even the most common daily transactions such as Internet browsing, grocery shopping and online banking increase the exposure and vulnerability of threats to information privacy. In response to these threats, and concerns surrounding data integrity, security, online privacy and confidentiality, professional services firms have begun to expand their service lines to include third-party enforcement programs. Seals, or other easily distinguishable symbols, are issued to enterprises whose privacy policies and procedures have been concluded to be in compliance with its governing board(s). The purposes of third-party enforcement programs are three-fold: (i) to build consumer trust; (ii) to educate subjects of their privacy preferences; and (iii) to develop a complaint resolution mechanism. Here are the three major procedures of enforcing privacy policies in an organization:

- Building consumer trust
  - There exists a natural conflict of interest between organization and subject.
  - Subject is absent from the development of privacy policy.
  - Independent review of an organization's compliance with governing board(s) adds reliability and credibility.
- Educate subjects of their privacy preferences
  - Promote awareness of privacy issues and how to get in contact with privacy coordinator.
  - If for any reason a subject's PII is required for another purpose from when it was collected, it is the enterprise's responsibility to obtain additional consent.
  - At a minimum, the enterprise is to inform the subject of the circumstance and provide an opportunity for the subject to opt out of such a use.
- Complaint resolution mechanism
  - Organizations rewarded a seal of validation are required to provide subjects a method to resolve any problems or discuss any complaints.
  - Complaint resolution process should be easily accessible and comprehendible.

In order to stand apart from industry rivals, companies strive to obtain a competitive advantage. As an organization, it is advantageous to be knowledgeable of the external risks associated affecting the protection of a subject's privacy. Participants of third-party enforcement programs are continuously monitored for adaptability to legislative frameworks and threats to privacy compliance.

Privacy technologies have been researched for a period of time. For example, the Platform for Privacy Preferences Project (P3P) working group at World Wide Web Consortium (W3C) develops the P3P

specification for enabling Web sites to express their privacy practices in a standard and machine-readable XML format. P3P user agents allow users to automatically be informed of site practices and to automate decision-making based on the Web sites' privacy practices. In addition, P3P also provides a language called P3P Preference Exchange Language 1.0 (APPEL1.0) that is used to express the user's preferences for making automated or semi-automated decisions regarding the acceptability of machine-readable privacy policies from P3P enabled Web sites. It provides a base schema for the data collected and a vocabulary to express purposes, the recipients, and the retention policy. Although it captures common elements of privacy policies, sites may have to provide further explanations in human-readable policies.

Furthermore, WS-Privacy has been mentioned in industry for a period of time for defining subject privacy preferences and organizational privacy practice statements for Web services. At this minute, the WS-Privacy specification has not been released to public yet. Then, the Enterprise Privacy Authorization Language (EPAL) technical specification is used to formalize privacy authorizations for actual enforcement within an intra- or inter- enterprise for business-to-business privacy control. On the other hand, the XACML is a general- purpose access control policy language used to describe policy and access control decision request/response [11]. Though XACML has drafted a privacy policy profile document [12], the current XACML framework can not handle the privacy enforcement.

One of the most significant privacy technologies is the IBM Tivoli Privacy Manager for e-business. This privacy middleware technology converts privacy policy and data-handling rules from applications and IT systems to P3P format. The future development will include the integration of privacy technologies into some specific Web-based applications like in healthcare sector (e.g., Microsoft Healthvault and Google Health Portal), especially in light of recent changes in health privacy legislative environment.

Cross-references

- ▶ Data Privacy and Patient Consent
- ▶ Privacy
- ▶ Privacy-Enhancing Technologies
- ▶ Privacy Metrics
- ▶ Privacy-Preserving Data Mining

Recommended Reading

1. Cheng V.S.Y. and Hung P.C.K. Health Insurance Portability and Accountability Act (HIPAA) compliant access control model for web services. IJHISI, 1(1):22–39, 2005.
2. Fischer-Hubner S. IT-security and privacy. Lecture notes in computer science, Springer, Berlin Heidelberg New York, 2001.
3. Online Privacy Alliance. Effective Enforcement of Self Regulation. Online: http://www.privacyalliance.org/resources/enforcement.shtml
4. Powers C.S., Ashley P., and Schunter M. Privacy promises, access control, and privacy management – enforcing privacy throughout an enterprise by extending access control. In Proc. Third Int. Symp. on Electronic Commerce, pp. 13–21. IEEE Computer Society, 2002.

Privacy Protection

▶ Statistical Disclosure Limitation For Data Access

Privacy-Enhancing Technologies

Simone Fischer-Hübner
Karlstad University, Karlstad, Sweden

Synonyms

PETs

Definition

Privacy-Enhancing Technologies (PETs) can be defined as technologies that are enforcing privacy principles in order to protect and enhance the privacy of users of information technology (IT) and/or of individuals about whom personal data are processed (the so-called data subjects). Privacy principles that PETs are enforcing can be derived from internationally acknowledged privacy guidelines or legislation, such as the OECD Privacy Guidelines [10] and the EU Data Protection Directive 95/46/EC [7]. One fundamental privacy principal that serves as the foundation for the Privacy-Enhancing Technologies that are aiming at providing *anonymity*, *pseudonymity*, or *unobservability* for users and/or other data subjects, is the privacy principles of data minimization. It requires that the collection of personally identifiable data should be minimized (and if possible avoided), because obviously privacy is best protected if no personal data at all (or at least as little data as possible) is collected or processed. Further

important privacy principles addressed by other PETs are the informed consent by data subjects (as a prerequisite for making data processing legitimate), transparency (i.e., openness) of data processing, and appropriate technical security means for protecting the confidentiality, integrity and availability personal data.

Historical Background

IT security technologies for protecting the confidentiality, integrity and availability of (personal) data, such as *access control*, have been developed and researched since the beginning of computing – *data encryption* as a technique for protecting the confidentiality of data has been used since ancient times. *Inference controls* for protecting the data of individuals stored in statistical databases (e.g., medical databases) have been elaborated since the 1970's.

Most of the fundamental anonymity technology concepts were introduced by David Chaum in the 1980's (see: [3,4,2]). The term "Privacy-Enhancing Technologies" was first introduced in 1995 in a report on data minimization technologies, which was jointly published by the Dutch Registratiekamer and the Information and Privacy Commissioner in Ontario with the title "Privacy-Enhancing Technologies: The path to Anonymity" [13]. Since then, further research and development has been done in data minimization technologies (e.g., [12,14]), and areas such as privacy policies [9,10] and privacy enhancing identity management [11].

Privacy as an expression of the rights of self-determination and human dignity is considered a core value in democratic societies and is recognized either explicitly or implicitly as a fundamental human right by most constitutions of democratic societies. However, in the network society, individuals are increasingly at risk that all their communications, transactions and movements can be monitored and profiled. Profiles collected at various sites can be easily combined, aggregated and retained without time limitations and without the individuals' knowledge or consent. Research and development of PETs have been motivated by the vision to provide technical means allowing individuals to retain control over their personal spheres, i.e., to protect their privacy in the electronic information age.

Foundations

Privacy-Enhancing Technologies can basically be divided into:

1. The class of PETs for minimizing or avoiding identifiable data for users and/or other data subjects.
2. The class of PETs for safeguarding lawful and privacy-friendly personal data processing.
3. PETs that are a combination of the two aforementioned PET classes.

PETs for Minimizing or Avoiding Personally Identifiable Data

To the class PETs that are minimizing or eliminating personally identifiable data of data subjects (that are not acting as users) belong *inference controls* in statistical databases. In general-purpose database systems (e.g., medical databases), some users (e.g., physicians) need to access personal data attributes, while other user (e.g., researchers) only need to access a personal database by statistical queries. Thus, access control mechanisms should allow certain users to access data in anonymous form by performing only statistical queries. However, by correlating different statistics (i.e., by launching so-called tracker attacks), a user may succeed in deducing confidential information about some individual. *Inference controls* in statistical databases shall ensure that the statistics released by the database do not lead to the disclosure of any confidential personal data.

PETs that are minimizing/avoiding identifiable data of users, and that are thereby providing *anonymity*, *pseudonymity*, and/or *unobservability*, can be divided dependent on whether data is minimized on communication level or on application level. Examples for anonymous communication schemes for achieving sender anonymity are DC-nets [4], Mix-nets [2] or Crowds [14], where the latter two ones will be presented in more detail below.

With DC-nets, the fact that someone is sending a message is hidden by a one-time pad encryption, which means that DC-nets can offer *perfect sender anonymity* (in the information-theoretic sense). By the use of message broadcast and implicit addresses (i.e., by the use of an attribute which allows only the addressee to recognize that the message is addressed to him, e.g., the message is public-key encrypted and the addressee is the only one who can successfully decrypt it with his secret key), it also provides *receiver and relationship anonymity*.

Another example for a protocol providing receiver anonymity is private information retrieval [2], which

assures that a powerful attacker is not able to find out what information another user has requested (i.e., who has received certain information). The goal is to request exactly one datum that is stored in a remote memory cell of a server without revealing which datum is requested.

Mix Nets The technique of a Mix network, which was originally introduced by David Chaum [4], realizes unlinkability of a sender and recipient (*relationship anonymity*), as well as, *sender anonymity* against the recipient, and optionally *recipient anonymity* (via so-called anonymous return addresses, which is however not elaborated further here).

A mix is a special network station, which collects and stores incoming messages, discards repeats, changes their appearance by encryption, and outputs them in a different order, and by this hides the relation between incoming and outgoing messages. If a sender of a message uses one mix for forwarding a message to a recipient on his behalf, the relation between sender and recipient is hidden from everybody but the mix and the sender of the message. This means also that the recipient only learns that the message was sent to him by the mix, but he does not know the identity of the real sender. However, if only one mix is used, this mix can in detail learn and potentially profile who is communicating with whom. To improve security, a message is sent over a mix net, which consists of a chain of independent mixes. The sender must perform cryptographic operations inverse to those of the mix, because the recipient must be able to read the message. A global attacker, who can monitor all communication lines, should in principle only be able to trace a message through the mix network, if he has the cooperation of all mix nodes on the path or if he can break the cryptographic operations. Thus, in order to ensure unlinkability of sender and recipient, at least one mix in the chain has to be trustworthy.

Assume that Alice wants to send anonymously a message msg to recipient Bob (which could be encrypted with Bob's public key for providing also message secrecy). Alice chooses a mix sequence Mix_1, Mix_2,..., Mix_m (In Fig. 1, Alice chooses m = 3 Mixes). Let for simplicity Mix_{m+1} denote the recipient (Bob). Each Mix_i with address A_i has initially chosen a key pair (c_i, d_i), where c_i is a public key of Mix_i and d_i is its private key. Let z_i be a random string.

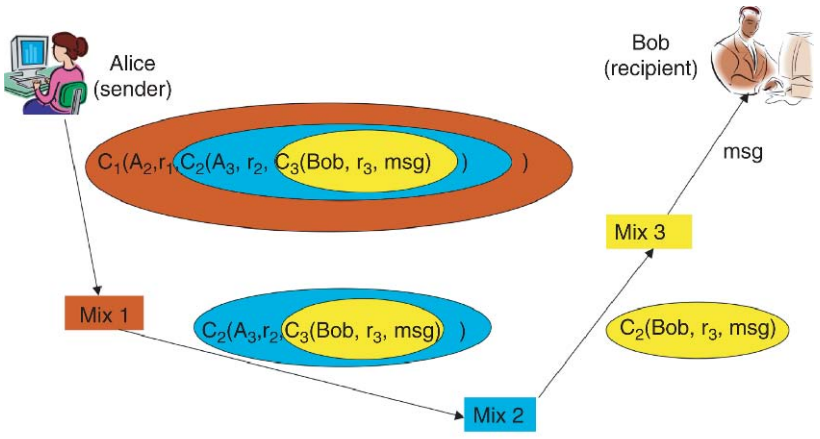
Alice recursively creates the following encrypted messages, where M_i is the message that Mix_i will receive:

$$M_{m+1} = msg$$

$$M_i = c_i(z_i, A_{i+1}, M_{i+1}) \quad \text{for } i = 1,\ldots, m,$$

i.e., she adds one layer of public key encryption to the message in reverse order to the sequence of mixes in the chain. She then sends the result M_1 to Mix_1.

Each Mix_i decrypts the incoming message $M_i = c_i(z_i, A_{i+1}, M_{i+1})$ with its private key d_i, discards the random string z_i, and finds the address A_{i+1} of the next mix Mix_{i+1} in the chain and the message M_{i+1}, which it forwards to Mix_{i+1}. The random string z_i is needed, because otherwise an attacker could again, by



Privacy-Enhancing Technologies. Figure 1. Mix-network with a chain of three mixes.

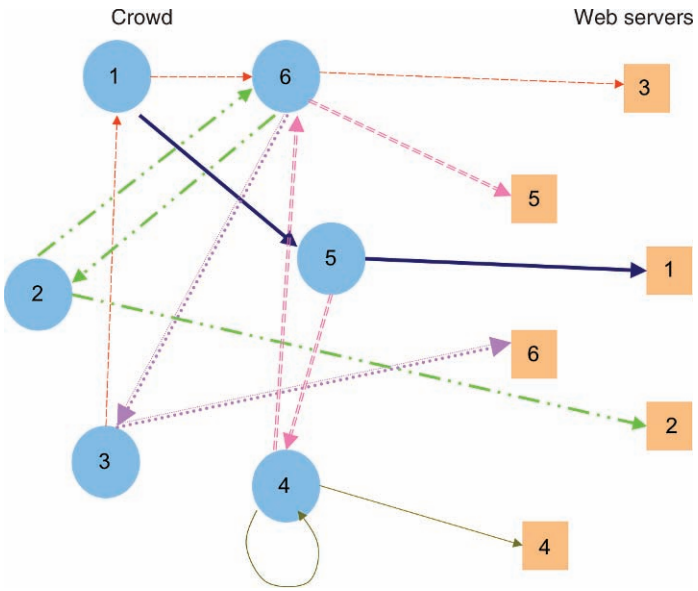
encrypting an outgoing message from a mix with the public key of the mix and comparing the result with the former inputs to the mix, succeed to relate an input to its output (i.e., he would be able to trace the message flow through the mix). Besides *anonymity* protection, mix nets can also provide *unobservability*, if dummy messages (i.e., meaningless messages) are sent out by the participants in order to hide information about if and when meaningful messages were sent.

Attacks on mix nets and possible countermeasures have been discussed in [8].

Mix-nets have first been developed for anonymous email applications. Later the Mix-net concept has been used for establishing anonymous bi-directional real-time communication. One example for distributed overlay networks designed to anonymize real-time, bidirectional TCP-based communications is Onion Routing, in which the sender's proxy constructs an onion which encapsulates a route to the responder (e.g., Web server). An Onion is an object with layers of public key encryption, which is used to produce anonymous bi-directional virtual circuit between communication partners (in Mix nets layered public key encryption are also used to build up a path) and to distribute symmetric keys to the nodes on the path. The sender's proxy constructs an onion which encapsulates a route to the responder. Faster symmetric encryption with the symmetric keys that were distributed to the nodes on the path with the help of the Onion is used for data communication via the virtual path once it has been established.

Crowds Reiter and Rubin developed Crowds – a system based on the idea that users can make anonymous Web transactions when they blend into a "crowd" [14]. A crowd is a geographically diverse group that performs Web transactions on behalf of its members. Each crowd member runs a process on his local machine called "jondo." Once started, the jondo engages in a protocol to join the crowd, during which it is informed of the other current crowd members and in which the other crowd members are informed of the new jondo's membership. Besides, the user configures his browser to employ the local jondo as a proxy for all network services. Thus all http requests are directly sent to the jondo rather than to the end Web-server and the jondo forwards the request to a randomly chosen crowd member. Whenever a crowd member receives a request from another jondo in the crowd, it makes a random choice to forward the request to another crowd member with a probability $p_f > ½$ or to submit the request to the end Web server to which the request was destined. Figure 2 shows a possible set of virtual paths that are created according to the Crowds protocol.

The server's reply is sent backward along the path, with each jondo sending it to its predecessor on the



Privacy-Enhancing Technologies. Figure 2. Paths in a crowd, where for each path the initiator (sender) and web server are labeled the same [14].

path. All communication between jondos is encrypted by a cryptographic key shared between the jondos involved.

Crowds provides sender anonymity against the end Web-server, since the end server obtains no information regarding who initiated any given request (each Crowds member is equally "suspicious"). Second, since a jondo on the path cannot distinguish whether its predecessor on the path initiated the request or is forwarding it, no jondo on the path can learn the initiator of a request. Since all communication between jondos is encrypted, Crowds also offers receiver anonymity against a local eavesdropper (e.g., local gateway administrator) that can observe the communication involving the user's machine unless the originator of the request ends up submitting the request itself. (However, the probability that an originator submits its own request decreases as the crowd size increases).

Crowds enables very efficient implementations that typically outperforms mixes that uses layered encryption techniques. However, in contrast to mix-nets, crowds cannot protect against global attackers.

PETs for anonymous or pseudonymous applications/transactions can be implemented on top of anonymous communication protocols. Examples are anonymous E-cash protocols based on *blind signatures* [3], as well as anonymous credential systems (such as Idemix [1]), which can be used to implement for instance anonymous or pseudonymous access control, e-health or e-government, and other anonymous or pseudonymous applications.

PETs for Safeguarding Lawful and Privacy-Friendly Personal Data Processing

It is not always possible to avoid the processing of personal data. Public authorities, health care providers, employers are example of organizations that still need to process personal data about their citizens, patients, employers for various legitimate reasons. If personal data are collected and processed, legal privacy requirements need to be fulfilled. The class of PETs described in this section comprises technologies that enforce legal privacy requirements in order to safeguard the lawful processing of personal data. The driving principles behind these types of PETs can also be found in data protection legislation, such as the EU Data Protection Directive 95/46/EC, which requires that controllers implement security measures which are appropriate to the risks presented for personal data in storage or transmission, with a view to protecting personal data against accidental loss, alteration, unauthorized access and against all other unlawful forms of processing.

Examples of technologies belonging to this class of PETs include classical security technologies, such as *data encryption* or *access control*, which protects the confidentiality and integrity of personal data. Other examples are technologies for stating or enforcing privacy policies: The Platform for Privacy Preferences Protocol (P3P) [6] increases transparency for the end users by informing them about the privacy policies of web sites and can hence be used to enforce the legal principles and requirement (e.g., according to Art.10 EU Data Protection Directive 95/46/EC) to inform data subjects about the purpose of data processing, identity of the controller, retention period, etc. Privacy policy models (such as the privacy model presented in [8]) can technically enforce privacy requirements such as purpose binding, and privacy policy languages, such as the Enterprise Privacy Authorization Language EPAL [9], can be used to encode and to enforce more complex enterprise privacy policies within and across organizations.

PETs that are a Combination of the Two Aforementioned PET Classes

The third class of PETs comprises technologies that are combining PETs of class 1 and class 2. An example is provided by privacy-enhancing identity management technologies as the ones that have been developed within the EU FP6 project PRIME (Privacy and Identity Management for Europe) [11]. Identity Management (IDM) subsumes all functionalities that support the use of multiple identities by the identity owner (user-side IDM) and by those parties with whom the owner interacts (services-side IDM). The PRIME project addresses privacy-enhancing IDM to support strong privacy by particularly avoiding or reducing identification and by technically enforcing informational self-determination.

PRIME is based on the principle that design must start from maximum privacy. Therefore, with the help of anonymous communication technologies and anonymous credential protocols, a priori all interactions are anonymous, and individuals can chose pseudonyms to link different interactions to each other, bind attributes and capabilities to pseudonyms and can establish end-to-end secure channels between pseudonyms. Whether or not interactions are linked to each other or to a certain pseudonym is under the

individual's control. For this, PRIME tools allow individuals to act under different pseudonyms with respect to communication partners, roles or activities. Policy management tools are helping them to define and negotiate privacy policies with services sides regulating who has the right to do what with one's personal data under which circumstances. Those policies are enforced at the receiving end, and there is enough evidence, so that users can actually trust in this enforcement. Transparency tools allow users to be informed about who has received what personal data related to them, and to trace personal data being passed on, and include online functions for exercising their rights to object to data processing or to rectify, block, delete data (see also [11] for more information).

Key Applications

As illustrated above, PETs can be applied in all kinds of application areas of IT, in which personal data is or can be collected or processed.

Cross-references

- ▶ Access Control
- ▶ Anonymity
- ▶ Blind Signatures
- ▶ Data Encryption
- ▶ Data Security
- ▶ Inference Control in Statistical Databases
- ▶ Privacy Policies and Preferences
- ▶ Pseudonymity
- ▶ Unobservability

Recommended Reading

1. Camenisch J. and van Herreweghen E. Design and implementation of the idemix anonymous credential system. In Proc. Ninth ACM Conf. on Computer and Communications Security, 2002, pp. 21–30.
2. Chaum D.L. Untraceable electronic mail, return addresses, and digital pseudonyms. Commun. ACM, 24(2):84–88, 1981.
3. Chaum D.L. Security without identification: card computers to make big brother obsolete. Informatik-Spektrum, 10:262–277, 1987.
4. Chaum D.L. The dining cryptographers problem: unconditional sender and recipient untraceability. J. Cryptol., 1(1):65–75, 1988.
5. Cooper D.A. and Birman K.P. Preserving privacy in a network of mobile computers. In Proc. IEEE Symp. on Security and Privacy, 1995, pp. 26–83.
6. Cranor L. Web Privacy with P3P. O'Reilly, Sebastopol, CA, 2002.
7. European Union. Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data. Official Journal L, 281:1995.
8. Fischer-Hübner S. IT-Security and Privacy: Design and Use of Privacy Enhancing Security Mechanisms. LNCS, vol. 1958, Springer, Berlin, 2001.
9. Karjoth G., Schunter M., and Waidner M. Platform for enterprise privacy practices: privacy-enabled management of customer data. In Proc. Second Workshop on Privacy Enhancing Technologies, 2002, pp. 69–84.
10. Organization for Economic Co-operation and Development. Guidelines on the protection of privacy and transborder flows of personal data. OECD Guidelines, 1980.
11. PRIME EU FP6 Project. Privacy and Identity Management for Europe. Available at: http://www.prime-project.eu/.
12. Reed M.G., Syverson P.F., and Goldschlag D.M. Anonymous connections and onion routing. IEEE J. Select. Areas Commn., 16(4):482–494, 1998.
13. Registratiekamer, Information and Privacy Commissioner/Ontario. Privacy-enhancing technologies: the path to anonymity. Achtergrondstudies en Verkenningen 5B, vols. I and II, Rijswijk, 1995.
14. Reiter M. and Rubin A. Anonymous web transactions with crowds. Commun. ACM, 42(2):32–48, 1999.

PETs

▶ Privacy-Enhancing Technologies

Privacy-Preserving Data Mining

Chris Clifton
Purdue University, West Lafayette, IN, USA

Definition

Data Mining techniques that use specialized approaches to protect against the disclosure of private information may involve anonymizing private data, distorting sensitive values, encrypting data, or other means to ensure that sensitive data is protected.

Historical Background

The field of privacy-preserving data mining began in 2000 with two papers of that name[1,4]. Both papers addressed construction of decision trees, approximating the ID3 algorithm while limiting disclosure of data. While the problems appeared similar on the surface, the fundamental difference in privacy constraints shows the complexity of this field. In [1], the

assumption was that individuals were providing their own data to a common server, and added noise to sensitive values to protect privacy. The key to the technique was to discover the original distribution of the data, enabling successful construction of the decision tree. In [4], the data was presumed to be divided between two (or a small number) of parties, and cryptographic *Secure Multiparty Computation* techniques are used to construct the decision tree without any site disclosing the actual data values it holds.

A third approach to privacy-preserving data mining, *Data Transformation*, was introduced in [5]. The idea behind this approach is to transform the data space (e.g., using techniques such as Random Projection) so that data items can no longer be tied to individuals, but sufficient information is preserved to enable data mining.

Research in the field has largely been directed toward:

- Supporting more data mining tasks and algorithms (e.g., clustering, classification, outlier detection)
- Different privacy constraints (e.g., *vertically partitioned* and *horizontally partitioned* data)

Techniques provide varying levels of privacy guarantees, data mining result accuracy, and run-time performance.

Foundations

There are several factors that must be identified in developing or choosing a privacy-preserving data mining approach. The first is the source of and privacy constraints on the data. In some cases, data is held by a relatively small number of trusted parties, such as credit reporting bureaus, insurance companies, or government agencies. The challenge is that while these parties are allowed to use the data for their own purposes, privacy constraints prevent them from disclosing the data to others.

Secure Multiparty Computation Methods address this problem, allowing a data mining model to be constructed from the distributed data using cryptographic approaches that prevent disclosure of the individual data items between the parties. The key components of such a method are the algorithm, which uses cryptographic computations to duplicate a distributed data mining algorithm without disclosing data from any site, and a proof that disclosure is controlled. Typically such proofs use a simulation argument, showing that given the final result, a party can simulate the messages received in running the protocol (thus showing that no valuable information was disclosed other than that inherent in the result.)

When data is more widely distributed, such as individuals providing their data to a server for use in data mining, *Data Transformation Methods* and *Data Perturbation* come into play. Data transformation methods modify the data so that the original values are lost, but important information (e.g., distances) is preserved. A typical example of transformation is dimensionality reduction, for example random projection [3]. Typically the providers of data must agree on a transformation, then transform their data and send it to a central data collector / data miner. Details of the transformation are kept secret from the data miner. In many cases, the transformation is individualized or constructed collaboratively in a way that protects against collusion. Such techniques require specialized versions of data mining algorithms to mine the transformed data. The techniques also involve a proof; in this case, showing the difficulty/impossibility of reversing the transformation, or the amount of background knowledge required to do so.

Data Perturbation is similar in that individuals modify their data before providing it to a central data warehouse. The differences is that data perturbation techniques add noise, while preserving the structure of the original data. The key to these techniques is specialized data mining algorithms that can utilize knowledge of the distribution of the noise, along with the noisy data, to obtain better results than mining on the noisy data alone. These approaches typically reconstruct the data *distribution*, then use this to guide the data mining, rather than mining the data directly. As the goal of most data mining is to obtain models that generalize well, models consistent with the distribution of the data are likely to have good accuracy. In addition to the reconstruction algorithm, data perturbation methods need a measure of the privacy provided; as data is not completely hidden, it is important to be able to measure the tradeoff between privacy and model accuracy.

Once the data privacy constraints are understood and a general approach is chosen, the next step is to develop an algorithm for the data mining model to be built. Privacy-preserving data mining algorithms typically replicate the results of traditional data mining algorithms, and have been developed for many of the standard machine learning approaches.

Key Applications

Privacy-preserving data mining has application wherever data mining is applied to data about individuals. Examples include recommender systems, medical studies, intelligence, and social network analysis. In addition, this technology can be useful for sharing sensitive corporate information, for example in supply chain management [2].

As an example, Figure 1 shows a scenario evaluating an Algorithm developed to identify terrorist organization financial patterns. To evaluate this, it is necessary to compare real financial records to see how many are identified by the algorithm. An outlier detection algorithm could be used to determine if the space identified by the algorithm reflects a cluster of "normal" financial patterns or just a few outliers, but accessing real financial records is problematic. Using the secure multiparty computation technique in [6], it is possible to identify the number of items in the vicinity of "Tom Terrorist" without revealing either the identity or specific values of any of the items. This gives us the result sought without exposing private data.

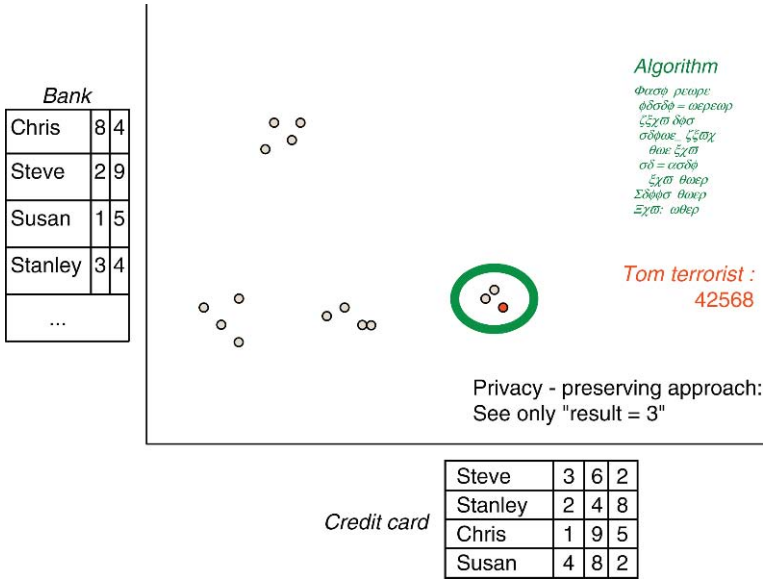
Future Directions

The main factors limiting adoption of this technology are:

1. The difficulty of doing exploratory data analysis. Privacy-preserving data mining techniques require that the data mining task to be performed be known without seeing the data.
2. Privacy standards and definitions. Current privacy regulations do not make it clear if this technology is either necessary or sufficient. Further work is needed to establish technically meaningful definitions of privacy.
3. Cost/performance. Many of these techniques are expensive in either computational power, their effect on result quality, and perhaps most important the cost of implementing a solution for a particular task.

Ongoing work is addressing these issues. Applications in corporate collaboration using corporate-sensitive data is likely to lead commercialization, as cost/benefit tradeoffs are easier to evaluate than with personal private data.

The field is closely related to *inference control in statistical databases* and *statistical disclosure limitation*; these fields have extensively studied the privacy risks inherent in releasing data under various types of perturbation and transformation. The key difference with privacy-preserving data mining is that the specific use of the data is assumed to be known. This allows methods that drastically distort the data, keeping only specific information intact rather than trying to preserve general statistical properties. The success of statistical disclosure limitation provides a good historical



Privacy-Preserving Data Mining. Figure 1. Using a privacy-preserving outlier detection algorithm to determine if an algorithm isolates terrorist behavior.

framework for the adoption of privacy-preserving data mining.

Cross-references

- ▶ Inference Control in Statistical Databases
- ▶ Privacy Metrics
- ▶ Randomized Methods to ensure Data Privacy
- ▶ Secure multiparty Computation Methods
- ▶ Statistical Disclosure Limitation for Data Access

Recommended Reading

1. Agrawal R. and Srikant R. Privacy-preserving data mining. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 439–450
2. Atallah M.J., Elmongui H.G., Deshpande V., and Schwarz L.B., Secure supply-chain protocols. In Proc. IEEE Int. Conf. on E-Commerce, 2003, pp. 293–302.
3. Kaski S. Dimensionality reduction by random mapping. In Proc. Int. Joint Conference on Neural Networks, 1999, pp. 413–418.
4. Lindell Y. and Pinkas B. Privacy preserving data mining. In Advances in Cryptology – CRYPTO 2000. Springer, 2000, pp. 36–54.
5. Oliveira S.R.M. and Zaïane O.R. Privacy preserving clustering by data transformation. In Proc. 18th Brazilian Symp. on Databases, 2003.
6. Vaidya J. and Clifton C. Privacy-preserving outlier detection. In Proc. 2004 IEEE Int. Conf. on Data Mining, 2004, pp. 233–240.
7. Vaidya J., Clifton C., and Zhu M. Privacy Preserving Data Mining. ser. Springer, Berlin, 2006.

Privacy-Preserving Spatial Queries

- ▶ Spatial Anonymity

Probabilistic Analysis

- ▶ Performance Analysis of Transaction Processing Systems

Probabilistic Data

- ▶ Data Uncertainty Management in Sensor Networks

Probabilistic Databases

Dan Suciu
University of Washington, Seattle, WA, USA

Synonyms

Uncertain databases

Definition

A probabilistic database is a database in which every tuple t belongs to the database with some probability $\mathbf{P}(t)$; when $\mathbf{P}(t) = 1$ then the tuple is certain to belong to the database; when $0 < \mathbf{P}(t) < 1$ then it belongs to the database only with some probability; when $\mathbf{P}(t) = 0$ then the tuple is certain not to belong to the database, and it is not necessary to bother representing it. A traditional (deterministic) database corresponds to the case when $\mathbf{P}(t) = 1$ for all tuples t. Tuples with $\mathbf{P}(t) > 0$ are called *possible tuples*. In addition to indicating the probabilities for all tuples, a probabilistic database must also indicate somehow how the tuples are correlated. In the simplest cases the tuples are declared to be either independent (when $\mathbf{P}(t_1t_2) = \mathbf{P}(t_1t_2)$), or exclusive (or disjoint, when $\mathbf{P}(t_1t_2) = 0$).

Historical Background

Extensions of databases to handle probabilistic data have been considered since the 1980s. In 1992, Barbara, Garcia-Molina, and Porter [6] developed a probabilistic data model that is an extension of the relational data model, in which relations have deterministic keys, and non-key attributes that are stohastic. They also introduced probabilistic analogs to the relational algebraic operators, later called *extensional operators*. Queries were restricted to return all deterministic keys of relations mentioned in the query body. Efforts to remove this restriction on queries lead to several extensions. In 1997, Lakshmanan, Leone, Ross, and Subrahmanian [19] modified the query semantics and proposed that probabilities be combined by user-defined *strategies*. In 1997, Fuhr and Roellke [17] proposed an alternative way to compute the correct semantics for all queries, using *intensional semantics*, by which every tuple in the query answer is annotated with a propositional formula called *event expression*: a general-purpose probabilistic inference algorithm can be used to compute the tuple's probability from the event expression, but this is known to be expensive in

general. Dalvi and Suciu in 2004 [12] showed that some queries can be computed efficiently using extensional operators if these are ordered carefully, to ensure that they are applied correctly: a plan consisting of correctly applied operators is called a *safe plan*. Not all queries admit a safe plan, but those that do not admit one can be shown to be #P-hard, thus they cannot be evaluated efficiently unless $P = NP$. Recent systems for managing probabilistic or incomplete databases include Trio [8,24], MystiQ [20], and MayBMS [4].

Foundations

For an illustration, consider the probabilistic database in Fig. 1. It has three tables, of which $\texttt{Product}^P$ and $\texttt{Customer}^P$ are probabilistic and $\texttt{Order}^P$ is deterministic. $\texttt{Product}^P$ contains three products; their names and their prices are known, but their color and shape are unknown. For example `Gizmo` may be have `color` = `red` and `shape` = `oval` with probability $p_1 = 0.25$, or may have `color` = `blue` and `shape` = `square` with probability $p_2 = 0.75$ respectively. `Camera` has three possible combinations of `color` and `shape`, and `IPod` has two. In other words, for every product in the database, the values for `color` and `shape`, are not known, but instead there is a probability space over their values. Each color-shape combination should exclude the others, so $p_1 + p_2 \leq 1$, $p_3 + p_4 + p_5 \leq 1$ and $p_6 + p_7 \leq 1$, holds for our table. When the sum is strictly less than one then that product may not occur in the table at all: for example `Camera` may be missing from the table with probability $1\text{-}p_3\text{-}p_4\text{-}p_5$.

Representation and Smantics

A popular representation of a probabilistic database consists of a regular database with two additional pieces of information: for each probabilistic table a distinguished attribute **P** is designated as the probability attribute, and a set of attributes is designated as *possible worlds key*. The key is interpreted as follows. If two tuples have the same values of the key attributes, then they are exclusive, i.e., only one of them may actually occur in the database. Otherwise, if the values of their key attributes differ then the tuples are independent. Tuples from different tables are always assumed to be independent. Figure 2a illustrates such a representation. Here the attribute **P** is designated as the probability attribute, and the two attributes `prod` and `price` are the possible worlds key. There are seven possible tuples, but at most three of them may actually occur in the database, because the first two are exclusive tuples, the next three are exclusive, and the last two are exclusive.

Tuples with the same possible worlds key are sometimes called *xor-tuples* or *x-tuples*. The entire set of such tuples is called a *maybe tuple*. The three `Camera`

$\texttt{Product}^P$

Prod	Price	Color	Shape	P
Gizmo	20	red	oval	$p_1 = 0.25$
		blue	square	$p_2 = 0.75$
Camera	80	green	oval	$p_3 = 0.3$
		red	round	$p_4 = 0.3$
		blue	oval	$p_5 = 0.2$
IPod	300	white	square	$p_6 = 0.8$
		black	square	$p_7 = 0.2$

Order

Prod	Price	Cust
Gizmo	20	Sue
Gizmo	80	Fred
IPod	300	Fred

$\texttt{Customer}^P$

Cust	City	P
Sue	NewYork	$q_1 = 0.5$
	Boston	$q_2 = 0.2$
	Seattle	$q_3 = 0.3$
Fred	Boston	$q_4 = 0.4$
	Seattle	$q_5 = 0.3$

Probabilistic Databases. Figure 1. A probabilistic database.

a

Product^P

Prod	Price	Color	Shape	P
Gizmo	20	red	oval	p_1
Gizmo	20	blue	square	p_2
Camera	80	green	oval	p_3
Camera	80	red	round	p_4
Camera	80	blue	oval	p_5
IPod	300	white	square	p_6
IPod	300	black	square	p_7

b

Product$_1$

Prod	Price	Color	Shape
Gizmo	20	red	oval
Camera	80	blue	oval
IPod	300	white	square

$\mathbf{P}(\text{Product}_1) = p_1p_5p_6$

Product$_2$

Prod	Price	Color	Shape
Gizmo	20	blue	square
Camera	80	blue	oval
IPod	300	white	square

$\mathbf{P}(\text{Product}_2) = p_2p_5p_6$

Product$_3$

Prod	Price	Color	Shape
Gizmo	20	red	oval
IPod	300	white	square

$\mathbf{P}(\text{Product}_3) = p_1(1-p_3-p_4-p_5)p_6$

Probabilistic Databases. Figure 2. A representation of a probabilistic table (a): it shows the probability of each tuple, and the key of the possible worlds: if two tuples share the same key values then they are exclusive, otherwise they are independent. Three possible worlds (b).

tuples in Fig. 2a are x-tuples, and together they form one maybe-tuple.

The semantics of a probabilistic database is a probability space over *possible worlds*. Product^P has 16 possible worlds, since there are two choices for Gizmo, four for Camera (including removing Camera altogether) and two for IPod. Figure 2b illustrates three of these sixteen possible worlds and their probabilities.

Query Evaluation

The semantics of a query q on a probabilistic database is defined as follows. Consider all possible worlds of the probabilistic database: each such world is a standard, deterministic database. Evaluate q on each possible world, using the standard semantics of a query on a deterministic database. Any tuple that is an answer to the query in some possible world is called a *possible answer tuple*. The *probability of a possible answer tuple* is defined as the sum of the probabilities of all worlds where the tuple is an answer. The semantics of the query is the set of possible answer tuples together with their probabilities.

A central problem in probabilistic databases is query evaluation. It is infeasible to enumerate all possible worlds, because their number is exponential is the size of the database. There are two approaches to query evaluation: extending relational algebra operators to manipulate probabilities explicitly, or using a general-purpose probabilistic inference algorithm. The first approach is rather similar to standard query processing, and is quite efficient: the declarative SQL query is translated into a relational algebra plan, called an *extensional plan*, then this plan is executed. We illustrate this below. However not all queries admit correct extensional plans. Those that do not admit such plans require a more expensive, general-purpose probabilistic inference, which have been studied in the context of knowledge representation systems and are not be discussed here.

The three main extensional operators on the three queries are illustrated in Fig. 3, then show how to combine them for more complex queries. The left column in the figure shows the queries in SQL syntax, the right column shows the same queries in datalog notation. In datalog, underline the variables that occur in the key positions. The first query, Q_1, asks for all the oval products in the database, and it returns two possible answer tuples:

prod	price	P
Gizmo	20	p_1
Camera	80	$p_3 + p_5$

For example the probability of the possible answer tuple Gizmo is the sum of the probabilities of the

```
Q1 : SELECT DISTINCT prod, price
     FROM Product^p
     WHERE shape = 'oval'

Q1(x,y) : - Product^p(x,y,'oval',z)

Q2 : SELECT DISTINCT city
     FROM Customer^p

Q2(y) : - Customer^p(x,y)

Q3 : SELECT DISTINCT*
     FROM Product^p, Order, Customer^p
     WHERE Product^p.prod = Order.prod
     and Product^p.price = Order.price
     and Order.cust = Customer^p.cust

Q3(*) : - Product^p(x,y,z),
          Order(x,y,u)
          Customer^p(u,v)
```

Probabilistic Databases. Figure 3. Three simple queries, expressed in SQL and in datalog.

8 possible worlds for Product (out of 16) where Gizmo appears as oval, and this turns out (after simplifications) to be p_1. These probabilities can be computed without enumerating all possible worlds, directly from the table in Fig. 2a by the following process: (i) Select all rows with shape='oval', (ii) project on prod, price, and **P** (the probability), (iii) eliminate duplicates, by replacing their probabilities with the sum, because they are disjoint events. The latter two steps are called a disjoint project:

Disjoint Project, $\pi_{\bar{A}}^{pD}$ If k tuples with probabilities $p_1,...,p_k$ have the same value, $\bar{a}$, for their $\bar{A}$ attributes, then the disjoint project will associated the tuple $\bar{a}$ with the probability $p_1 + ... + p_k$. The disjoint project is correctly applied if any two tuples that share the same values of the $\bar{A}$ attributes are disjoint events.

Q_1 can therefore be computed by the following plan:

$$Q_1 = \pi^{pD}_{\text{prod,price}}(\sigma_{\text{shape='oval'}}(\text{Product}^p))$$

$\pi^{pD}_{\text{prod,price}}$ is correctly applied, because any two tuples in Product^p that have the same prod and price are disjoint events.

Query Q_2 in Fig. 3 asks for all cities in the Customer table, and its answer is:

city	P
New York	q_1
Boston	$1 - (1 - q_2)(1 - q_4)$
Seattle	$1 - (1 - q_3)(1 - q_5)$

This answer can also be obtained by a projection with a duplicate elimination, but now the probabilities p_1, p_2, p_3,... of duplicate values are replaced with $1 - (1 - p_1)(1 - p_2)(1 - p_3)...$, since in this case all duplicate occurrences of the same city are independent. This is called an independent project:

Independent Project, $\pi_{\bar{A}}^{pI}$ If k tuples with probabilities $p_1,...,p_k$ have the same value, $\bar{a}$, for their $\bar{A}$ attributes, then the independent project will associated the tuple $\bar{a}$ with the probability $1 - (1 - p_1)(1 - p_2)...(1 - p_k)$. The independent project is correctly applied if any two tuples that share the same values of the $\bar{A}$ attributes are independent events.

Thus, the disjoint project and the independent project compute the same set of tuples, but with different probabilities: the former assumes disjoint probabilistic events, where $\mathbf{P}(t \vee t') = \mathbf{P}(t) + \mathbf{P}(t')$, while the second assumes independent probabilistic events, where $\mathbf{P}(t \vee t') = 1 - (1 - \mathbf{P}(t))(1 - \mathbf{P}(t'))$. Continuing our example, the following plan computes Q_2: $Q_2 = \pi^{pI}_{\text{city}}(\text{Customer}^p)$. Here π^{pI}_{city} is correctly applied because any two tuples in Customer^p that have the same city are independent events.

Query Q_3 in Fig. 3 illustrates the use of a join, and its answer is:

prod	price	color	shape	cust	city	P
Gizmo	20	red	oval	Sue	New York	p_1q_1
Gizmo	20	red	oval	Sue	Boston	p_1q_2
Gizmo	20	red	oval	Sue	Seattle	p_1q_3
Gizmo	20	blue	square	Sue	New York	p_2q_1
...	...					

It can be computed by modifying the join operator to multiply the probabilities of the input tables:

Join, $\bowtie^p$ Whenever it joins two tuples with probabilities p_1 and p_2, it sets the probability of the resulting tuple to be p_1p_2.

A plan for Q_3 is: Q_3 = Product $\bowtie^p$ Order $\bowtie^p$ Customer.

Extensional operators can be combined to form an extensional plan. A plan is called *safe* if each operator is applied correctly. It is possible to illustrate with the following probabilistic database schema:

$\text{Product}^p(\underline{\text{prod, price}}, \text{color, shape})$

$\text{Order}^p(\underline{\text{prod, price, cust}})$

Both tables are probabilistic. The following query finds all colors of products ordered by the customer named Sue:

$$Q_4(c) : -\text{Product}^p(\underline{x, y}, c, s), \text{Order}^p(\underline{x, y, Sue})$$

Q_4 admits the following safe plan:

$$Q_4 = \Pi^{pI}_{\text{color}}(\Pi^{pD}_{\text{prod,price,color}}(\text{Product}^p) \bowtie^p \sigma_{\text{cust}='\text{Sue}'}(\text{Order}^p))$$

This is safe because $\Pi^{pD}_{\text{prod,price,color}}$ combines only disjoint tuples, while Π^{pI}_{color} combines only independent tuples. Similarly, the join operators combines independent tuples, since the left operand consists of tuples combined from the Product^p table while the right operand consists of tuples from the order^p table.

In contrast, the following plan is not safe:

$$\Pi^{pI}_{\text{color}}(\text{Product}^p \bowtie^p \sigma_{\text{cust}='\text{Sue}'}(\text{Order}^p))$$

because the outermost projection Π^{I}_{color} will combine tuples that are not independent: there may be two or more distinct tuples in the join $\text{Product}^p \bowtie^p \text{Order}^p$ that have the same Product^p tuple, hence they are not independent.

Some queries do not admit any safe plans at all. Figure 4 illustrates three queries that are known not to admit any safe plan. All three are boolean queries, i.e., they return either "true" or nothing, but they still have a probabilistic semantics, and it is necessary to compute the probability of the answer "true." No combination of extensional operators result in a safe plan for any of these queries. For example $\pi^{pI}_{\emptyset}(\text{R} \bowtie \text{S} \bowtie \text{T})$ is an incorrect plan for H_1, because two distinct rows in R ⋈ S ⋈ T may share the same tuple in R, hence they are not necessarily independent events. In fact it has been shown that the complexity of each of these three queries is #P-hard, hence there is no polynomial time algorithm for computing their probabilities unless $P = NP$.

Key Applications

Probabilistic databases are design to allow uncertain data to be managed directly by a relational database system. Several types of applications have been considered.

In *Information Extraction* the goal is to extract structured data from a collection of unstructured text documents. Examples include address segmentation, citation extraction, extractions for comparison shopping, hotels, restaurant guides, etc. The schema is given in advance by the user, and the extractor is tailored to that specific schema. All approaches to extraction are imprecise, and most often can associate a probability score to each item extracted. A probabilistic database allows these probability scores to be stored natively.

In *Fuzzy Object Matching* the problem is to reconcile objects from two collections that have used different naming conventions. This is a central problem in data cleaning and data integration, and has also been called record linkage, or de-duplication. The basic approach is to compute some similarity score between pairs of objects, usually by starting from string similarity scores on their attributes then combining these

```
Schema: R(A),S(A,B), T(B)
H1: SELECT DISTINCT 'true' AS A
    FROM R, S, T
    WHERE R.A = S.A and S.B = T.B
```

$H_1 : - R(\underline{x}), S(\underline{x,y}), T(\underline{y})$

```
Schema: R(A,B), S(B)
H2: SELECT DISTINCT 'true' AS A
    FROM R, S
    WHERE R.B = S.B
```

$H_2 : - R(\underline{x,y}), S(\underline{y})$

```
Schema: R(A,B), S(A, B)
H3: SELECT DISTINCT 'true' AS A
    FROM R, S
    WHERE R.A = S.A and R.B = S.B
```

$H_3 : - R(\underline{x,y}), S(x,\underline{y})$

Probabilistic Databases. Figure 4. Three queries that do not admit safe plans, and that are known to be #P-complete.

scores into a global score for the pair of objects. Thus, the result of fuzzy object matching is inherently probabilistic. In traditional data cleaning it needs to be determinized somehow, e.g., by comparing the similarity score with a threshold and classifying each pair of object into a *match*, a *non-match*, and a *maybe-match*. A probabilistic database allows users to keep the similarity scores natively.

Other applications include: management of sensor data and of RFID data, probabilistic data integration, probabilistic schema matches, and flexible query interfaces.

Cross-references

- ► Data Uncertainty Management in Sensor Networks
- ► Incomplete information
- ► Inconsistent databases
- ► Uncertainty Management in Scientific Database Systems

Recommended Reading

1. Adar E. and Re C. Managing uncertainty in social networks. IEEE Data Eng. Bull., 30(2):15–22, 2007.
2. Andritsos P., Fuxman A., and Miller R.J. Clean answers over dirty databases. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
3. Antova L., Koch C., and Olteanu D. $10^{\wedge}(10^{\wedge}6)$ worlds and beyond: Efficient representation and processing of incomplete information. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
4. Antova L., Koch C., and Olteanu D. MayBMS: Managing incomplete information with probabilistic world-set decompositions (demonstration). In Proc. 23rd Int. Conf. on Data Engineering, 2007.
5. Antova L., Koch C., and Olteanu D. World-set decompositions: expressiveness and efficient algorithms. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 194–208.
6. Barbara D., Garcia-Molina H., and Porter D. The management of probabilistic data. IEEE Trans. Knowl. Data Eng., 4(5):487–502, 1992.
7. Benjelloun O., Das Sarma A., Halevy A., and Widom J. ULDBs: databases with uncertainty and lineage. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 953–964.
8. Benjelloun O., Das Sarma A., Hayworth C., and Widom J. An introduction to ULDBs and the Trio system. IEEE Data Eng. Bull., 29(1):5–16, 2006.
9. Cheng R., Kalashnikov D., and Prabhakar S. Evaluating probabilistic queries over imprecise data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 551–562.
10. Cheng R. and Prabhakar S. Managing uncertainty in sensor databases. ACM SIGMOD Rec., 32(4):41–46, December 2003.
11. Dalvi N., Re C., and Suciu D. Query evaluation on probabilistic databases. IEEE Data Eng. Bull., 29(1):25–31, 2006.
12. Dalvi N. and Suciu D. Efficient query evaluation on probabilistic databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
13. Dalvi N. and Suciu D. Management of probabilistic data: foundations and challenges. In Proc. 26th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2007, pp. 1–12. (invited talk).
14. Das Sarma A., Benjelloun O., Halevy A., and Widom J. Working models for uncertain data. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
15. Deshpande A., Guestrin C., Madden S., Hellerstein J.M., and Hong W. Model-driven data acquisition in sensor networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 588–599.
16. Deshpande A., Guestrin C., Madden S., Hellerstein J.M., and Hong W. Using probabilistic models for data management in acquisitional environments. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 317–328.
17. Fuhr N. and Roelleke T. A probabilistic relational algebra for the integration of information retrieval and database systems. ACM Trans. Inf. Syst., 15(1):32–66, 1997.
18. Gupta R. and Sarawagi S. Creating probabilistic databases from information extraction models. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 965–976.
19. Lakshmanan L., Leone N., Ross R., and Subrahmanian V.S. Probview: a flexible probabilistic database system. ACM Trans. Database Syst., 22(3), 1997.
20. MystiQ: a probabilistic database system, http://mystiq.cs.washington.edu/.
21. Re C. and Suciu D. Materialized views in probabilistic databases for information exchange and query optimization. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
22. Re C., Letchner J., Balazinska M., and Suciu D. Event queries on correlated probabilistic streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2008.
23. Re C. and Suciu D. Approximate lineage for probabilistic databases. In Proc. 34th Int. Conf. on Very Large Data Bases, 2008.
24. Widom J. Trio: A system for integrated management of data, accuracy, and lineage. In Proc. 2nd Biennial Conf. on Innovative Data Systems Research, 2005, pp. 262–276.

Probabilistic Model

- ► BM25
- ► Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model

Probabilistic Model of Indexing

- ► Two-Poisson model

Probabilistic Querying

- ► Data Uncertainty Management in Sensor Networks

Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model

Thomas Roelleke[1], Jun Wang[1], Stephen Robertson[2]
[1]Queen Mary University of London, London, UK
[2]Microsoft Research Cambridge, Cambridge, UK

Synonyms

BIR model; Probabilistic model; RSJ model

Definition

Information retrieval (IR) systems aim to retrieve relevant documents while not retrieving non-relevant ones. This can be viewed as the foundation and justification of the binary independence retrieval (BIR) model, which proposes to base the ranking of documents on the division of the probability of relevance and non-relevance.

For a set r of relevant documents, and a set $\bar{r}$ of non-relevant documents, the BIR model defines the following term weight and retrieval status value (RSV) for a document-query pair "d, q":

$$\text{birw}(t, r, \bar{r}) := \frac{P(t|r) \cdot P(\bar{t}|\bar{r})}{P(t|\bar{r}) \cdot P(\bar{t}|r)} \quad (1)$$

$$\text{RSV}_{\text{BIR}}(d, q, r, \bar{r}) := \sum_{t \in d \cap q} \log \text{birw}(t, r, \bar{r}) \quad (2)$$

Here, $P(t|r)$ is the probability that term t occurs in the relevant documents, and $P(t|\bar{r})$ is the respective probability for term t in non-relevant documents.

There are two ways to estimate the term probabilities: (I_1) $P(t|r) := \frac{r_t}{R}$ and $P(t|\bar{r}) := \frac{n_t}{N}$, or (I_2) $P(t|r) := \frac{r_t}{R}$ and $P(t|\bar{r}) := \frac{n_t - r_t}{N - R}$. R denotes the number of relevant documents, and N denotes the number of all documents; r_t denotes the number of relevant documents containing term t, and n_t denotes the respective number of all documents. I_1 assumes term independence in the relevant documents and in *all* documents; I_2 assumes term independence in the relevant documents and in the *non-relevant* documents.

Given the two independence assumptions, and given the option to consider term presence only, there are four forms/variations of the BIR term weight. Further, the BIR model proposes a technique for dealing with missing relevance, where a zero probability problem needs to be avoided. The definitions given next show the case referred to as F4 where I_2 is assumed and term absence is considered. The constant 0.5 caters for missing relevance data.

$$\text{birw}_{+0.5}(t, r, \bar{r}) := \frac{(r_t + 0.5)/(R - r_t + 0.5)}{(n_t - r_t + 0.5)/(N - R - (n_t - r_t) + 0.5)} \quad (3)$$

$$\text{RSV}_{\text{BIR, F4}}(d, q, r, \bar{r}) := \sum_{t \in d \cap q} \log \text{birw}_{+0.5}(t, r, \bar{r}) \quad (4)$$

The derivation of the term weight and the retrieval status value is to be found in the scientific fundamentals of this entry.

Historical Background

The paper [10] on "Relevance Weighting of Search Terms" proposed what became known as the probabilistic retrieval model, binary independence retrieval model, or RSJ (initials of the authors) model.

Closely related is the "The Probability Ranking Principle in IR" [8] to justify that a ranking based on the probability of relevance is an optimal ranking, since the ranking minimises the costs for browsing the ranking.

Reference [2] on "Using Probabilistic Models of Document Retrieval without Relevance Information" investigated the generalisation for missing relevance, and [4] contributed "An Evaluation of Feedback in Document Retrieval using Co-occurrence Data."

Foundations

Derivation of the BIR Model

Derivations of the BIR model can be found in [6], [3] (pp. 21–31), and [5] (pp. 167–175).

The BIR model is derived from the odds of the probability of relevance:

$$O(r|d, q) := \frac{P(r|d, q)}{P(\bar{r}|d, q)} = \frac{P(r|d, q)}{\text{I} - P(r|d, q)} \quad (5)$$

Here, $P(r|d, q)$ is the probability that the event relevant (r) occurs for a given document-query pair. $P(\bar{r}|d, q)$ is the respective probability for non-relevant.

The derivation of the BIR model can be represented in six steps:

1. Bayes theorem and reduction of $O(r|d, q)$.
2. Representation of d as a vector $\vec{x}$ of binary term features (the binary term features are "term does occur" and "term does not occur").
3. Independence assumption for features.
4. Split product over features into two products: a product for term presence, and a product for term absence.
5. Non-query term assumption: if non-query terms occur with the same frequency in relevant and non-relevant documents, then they do not affect the ranking.
6. Rewrite the expression to obtain a compact form.

Bayes Theorem To estimate the unknown probability $P(r|d, q)$, the theorem of Bayes is applied:

$$P(r|d,q) := \frac{P(r|d,q)}{P(d,q)} = \frac{P(d|r,q)\cdot P(r|q)\cdot P(q)}{P(d,q)} \quad (6)$$

For $O(r|d, q)$, the probabilities $P(q)$ and $P(d, q)$ drop out. With respect to the ranking of documents, the factor $\frac{P(r|q)}{P(\overline{r}|q)}$ has no effect, since it is document-independent. Thus, the ranking of documents is based on the following ranking equivalence:

$$O(r|d,q) \stackrel{rank}{=} \frac{P(d|r,q)}{P(d|\overline{r},q)} \quad (7)$$

Binary Feature Vector $\vec{x}$ The next step concerns the representation and decomposition of the document d. For this, the BIR model assumes that d is a vector of binary features; this vector is denoted $\vec{x} = (x_1,\ldots,x_n)$. Each x_i corresponds to a term t_i, and for all x_i, $x_i = 1$ if term is present, and $x_i = 0$ if term is absent.

Independence Assumption Next, the BIR model assumes that the binary features are independent. This leads to the following equation for $P(d|r, q)$:

$$P(d|r,q) = P(\vec{x}|r,q) = \prod_{x_i} P(x_i|r,q) \quad (8)$$

The equation for $P(d|\overline{r}, q)$ is accordingly.

Product Split The product over all features is split into two products: one product for term presence ($x_i = 1$), and one for absence ($x_i = 0$).

$$P(d|r,q) = \left[\prod_{x_i=1} P(x_i|r,q)\right]\cdot\left[\prod_{x_i=0} P(x_i|r,q)\right] \quad (9)$$

Since $x_i = 1$ corresponds to "term occurs," and $x_i = 0$ corresponds to "term does not occur," the next equation replaces x_i by the term event t, where t means "term occurs."

$$P(d|r,q) = \left[\prod_{t\in d} P(t|r,q)\right]\cdot\left[\prod_{t\notin d} P(\overline{t}|r,q)\right] \quad (10)$$

In classical literature [10,6], the symbols p_i and q_i are employed instead of $P(t|r, q)$ and $P(\overline{t}|r, q)$; however, since the probabilities are more explicit, and avoid a confusion of query (q) and absence of term (q_i), and facilitate to relate the BIR model to other retrieval modes, the probabilistic notion is chosen here, while keeping in mind that $P(t|r, q)$ refers to the binary feature of term t.

Non-Query Term Assumption For all non-query terms, it is assumed that their frequency in relevant documents is equal to their frequency in non-relevant documents, i.e., $\forall t \notin q : P(t|r, q) = P(t|\overline{r}, q)$. This leads to a reduction of the product, i.e., the non-query terms are dropped.

$$\frac{P(d|r,q)}{P(d|\overline{r},q)} = \left[\prod_{t\in d\cap q}\frac{P(t|r,q)}{P(t|\overline{r},q)}\right]\cdot\left[\prod_{t\in q\setminus d}\frac{P(\overline{t}|r,q)}{P(\overline{t}|\overline{r},q)}\right] \quad (11)$$

This assumption corresponds to $\frac{P(t|r,q)}{P(t|\overline{r},q)} = 1$, and the more general assumption is $\frac{P(t|r,q)}{P(t|\overline{r},q)} = c$, where c is a constant.

Rewriting to Achieve Compact Form Finally, a rewriting leads to a compact form. The rewriting is based on the following equation:

$$1.0 = \prod_{t\in d\cap q}\left[\frac{P(\overline{t}|\overline{r},q)}{P(\overline{t}|r,q)}\cdot\frac{P(\overline{t}|r,q)}{P(\overline{t}|\overline{r},q)}\right] \quad (12)$$

Equation 11 is multiplied with this 1.0. This fills up the right product over query-only terms to a product over all query terms; then, this product is document-independent, and it can be dropped for ranking purpose, as the next equations illustrate.

$$O(r|d,q) = \left[\prod_{t\in d\cap q}\frac{P(t|r,q)}{P(t|\bar{r},q)}\right]\cdot\left[\prod_{t\in q\setminus d}\frac{P(\bar{t}|r,q)}{P(\bar{t}|\bar{r},q)}\right]\cdot \left[\prod_{t\in d\cap q}\frac{P(\bar{t}|\bar{r},q)}{P(\bar{t}|r,q)}\cdot\frac{P(\bar{t}|r,q)}{P(\bar{t}|\bar{r},q)}\right] \stackrel{rank}{=} \prod_{t\in d\cap q}\frac{P(t|r,q)\cdot P(\bar{t}|\bar{r},q)}{P(t|\bar{r},q)\cdot P(\bar{t}|r,q)} \tag{13}$$

The fractional expression is referred to as the BIR term weight, where either the above form or the logarithm of it may be viewed as the term weight. The following definition follows the first approach, and the logarithm is applied to the term weight in the definition of $\mathrm{RSV}_{\mathrm{BIR}}$.

$$\mathrm{birw}(t,r,\bar{r}) := \frac{P(t|r,q)\cdot P(\bar{t}|\bar{r},q)}{P(t|\bar{r},q)\cdot P(\bar{t}|r,q)} \tag{14}$$

$$\mathrm{RSV}_{\mathrm{BIR}}(d,q,r,\bar{r}) := \sum_{t\in d\cap q}\log \mathrm{birw}(t,r,\bar{r}) \tag{15}$$

The BIR weight is greater than 1.0 for *good* terms, i.e., good terms occur more frequently in relevant documents than in non-relevant documents. For $P(t|r, q) = P(t|\bar{r}, q)$, the BIR weight is 1.0; thus, such terms have no effect on the ranking. For *poor terms*, the BIR weight is less than 1.0; thus, poor terms have a negative effect on the $\mathrm{RSV}_{\mathrm{BIR}}$.

Alternative Derivation An alternative, shorter derivation is shown next. The abbreviations $p_i := P(x_i = 1|r)$ and $q_i := P(x_i = 1|\bar{r})$ are applied, and the document probabilities are expressed as follows:

$$\frac{P(d|r)}{P(d|\bar{r})} = \prod_i \frac{p_i^{x_i}\cdot(1-p_i)^{1-x_i}}{q_i^{x_i}\cdot(1-q_i)^{1-x_i}} \tag{16}$$

The exponent $x_i = 1$ selects the probability p_i that the term occurs, and $x_i = 0$ selects the probability $(1 - p_i)$ that the term does not occur. Resolving the exponent $1 - x_i$ leads to the next equation and ranking equivalence.

$$\frac{P(d|r)}{P(d|\bar{r})} = \left[\prod_i\left(\frac{p_i}{1-p_i}\cdot\frac{1-q_i}{q_i}\right)^{x_i}\right]\cdot\prod_i\frac{1-p_i}{1-q_i} \tag{17}$$

$$\stackrel{rank}{=} \prod_{t\in d\cap q}\mathrm{birw}(t,r,\bar{r}) \tag{18}$$

The right product ($\prod_i \frac{1-p_i}{1-q_i}$ in 17) is constant, and therefore does not affect the ranking. Since $x_i = 0$ for non-document terms, the first product reduces to $x_i = 1$ in d. With $p_i = q_i$ for non-query terms, the product reduces to $x_i = 1$ in d and q, i.e., the product is over $t \in d \cap q$.

Estimation of Term Probabilities

For describing the estimation of term probabilities, the following notation is employed:

Traditional notation	Event-based notation	Comment
r_t	$n_D(t, r)$	Number of relevant documents with term t
R	$N_D(r)$	Number of relevant documents
n_t	$n_D(t, c)$	Number of documents with term t in the collection c
N	$N_D(c)$	Number of documents in the collection c
p_t	$P_D(t\|r)$	Term probability in relevant documents
q_t	$P_D(t\|\bar{r})$	term probability in non-relevant documents

The notation comprises the traditional symbols, and it shows an alternative, namely an event-based notation. The event-based notation is explicit regarding the event-space (D for documents). Therefore, it is applicable in a dual way to document-based and location-based event spaces. Also, events such as a document, a collection, and any set of documents (relevant, non-relevant, retrieved, all) can be referred to systematically.

The estimates of the term probabilities are given next:

Traditional notation	Event-based notation	Comment
$p_t = \frac{r_t}{R}$	$P_D(t\|r) = \frac{n_D(t,r)}{N_D(r)}$	Probability of term in relevant
$q_t = \begin{cases}\frac{n_t}{N} & I_1\\ \frac{n_t - r_t}{N-R} & I_2\end{cases}$	$P_D(t\|\bar{r}) = \begin{cases}\frac{n_D(t,c)}{N_D(c)} & I_1\\ \frac{n_D(t,c)-n_D(t,r)}{N_D(c)-N_D(r)} & I_2\end{cases}$	Probability of term in non-relevant assumptions I_1 and I_2

The I_2 assumption and the consideration of term absence lead to the following equation for the BIR term weight:

$$\text{birw}(t, r, \bar{r}) = \frac{r_t/(R - r_t)}{(n_t - r_t)/(N - R - (n_t - r_t))} \quad (19)$$

The next section groups the variations of the term weight.

Variations of the BIR Term Weight

[10] introduced four variations (forms) of the term weights: the four variations (referred to as F1, F2, F3, and F4) follow from the two estimates for $P(t|\bar{r})$, and whether or not term absence is considered.

Assumption	Term absence	Variation	Term Weight
I_1	no	F1	$\frac{r_t/R}{n_t/N}$
I_2	no	F2	$\frac{r_t/R}{(n_t-r_t)/(N-R)}$
I_1	yes	F3	$\frac{r_t/(R-r_t)}{n_t/(N-n_t)}$
I_2	yes	F4	$\frac{r_t/(R-r_t)}{(n_t-r_t)/(N-R-(n_t-r_t))}$

For further reading, the BIR term weights are summarised in [3], page 27. The variations form a comprehensive mathematical coverage; F4 is the prime choice.

Solving the Zero Probability Problem

The BIR term weight is not defined for missing relevance; $R = r_t = 0$ leads to a division by zero. Therefore, [10] proposes to add constants to the frequency counts:

$$\text{birw}_{+0.5}(t, r, \bar{r}) := \frac{(r_t + 0.5)/(R - r_t + 0.5)}{(n_t - r_t + 0.5)/(N - R - (n_t - r_t) + 0.5)} \quad (20)$$

The subscript +0.5 in $\text{birw}_{+0.5}(t, r, \bar{r})$ marks this term weight to be different from the bare term weight $\text{birw}(t, r, \bar{r})$ in 19; $\text{birw}_{+0.5}(t, r, \bar{r})$ is also referred to as $\text{rsj}(t, r, \bar{r})$. What is the explanation underlying this zero-probability "fix"?

To reach an explanation, insert $r_t + 0.5$ for each r_t, and insert $R + 1$ for each R. This corresponds to assuming that there is a *virtual* document that is relevant, and each term occurs in *half* of the relevant documents, i.e., $p_t = 0.5$. The virtual document is retrieved, i.e., $n_t + 1$. Finally, this requires to assume $N + 2$ documents, which corresponds to assuming that there are two virtual documents, one of which is relevant. The next equation illustrates the explanation via virtual documents:

$$\text{birw}_{+0.5}(t, r, \bar{r}) := \frac{(r_t + 0.5)/(R + 1 - (r_t + 0.5))}{((n_t + 1) - (r_t + 0.5))/((N + 2) - (R + 1) - ((n_t + 1) - (r_t + 0.5)))} \quad (21)$$

This expanded form reduces to 20.

From a more general point of view, the estimate $P(t|r) := \frac{r_t + k}{R + K}$ is applied to cover the case for missing relevance, where $\frac{k}{K} = 0.5$, and $k = 0.5$, and $K = 1$, for $\text{birw}_{+0.5}$. This formulation reminds of the Laplace law of succession. Since the notion of "half of a relevant document" has no intuition, the number of virtual documents could be scaled to four rather than two [13]. This leads to: $N + 4$, $n_t + 2$, $R + 2$, $r_t + 1$. Then, the BIR weight is as follows:

$$\text{birw}_{+1}(t, r, \bar{r}) := \frac{(r_t + 1)/(R - r_t + 1)}{(n_t - r_t + 1)/(N - R - (n_t - r_t) + 1)} \quad (22)$$

Relationship between the BIR Model, IDF, and BM25

The BIR model can be viewed as a theoretical argument to support IDF-based retrieval.

[2] proposes for missing relevance to assume that for all term, $P(t|r)$ is the same. This leads to a co-ordination level component in the retrieval function:

$$\sum_{t \in d \cap q} \log \frac{P(t|r)}{1 - P(t|r)} + \sum_{t \in d \cap q} -\log \frac{P(t|\bar{r})}{1 - P(t|\bar{r})}$$

The left component expresses the co-ordination level match. For large N and $N >> n_t$, the right component is similar to the IDF component. i.e., $\sum_{t \in d \cap q} -\log \frac{n_t}{N - n_t} \approx \sum_{t \in d \cap q} idf(t, c)$. [11] investigates how these assumptions are affected in the case of little relevance information.

[9] reviews the relationship of the BIR model and IDF, and theoretical arguments for IDF. The relationship between BIR and IDF is based on the definition $idf(t, c) := -\log P_D(t|c)$ and the estimation of the BIR probability $P(t|\bar{r})$. By assuming

$P(t|\bar{r}) = P_D(t|c)$, the relationship is established, i.e., $idf(t, c) = -\log P(t|\bar{r})$.

[13] builds on this relationship and shows a fully *idf*-based formulation of the BIR term weight:

$$\log \text{birw}(t, r, \bar{r}) = \log \frac{P(t|r) \,.\, P(\bar{t}|\bar{r})}{P(t|\bar{r}) \,.\, P(\bar{t}|r)} \quad (23)$$

$$= idf(t, \bar{r}) - idf(t, r) + idf(\bar{t}, r) - idf(\bar{t}, \bar{r}) \quad (24)$$

$$\approx idf(t, c) - idf(t, r) + idf(\bar{t}, r) - idf(\bar{t}, c) \quad (25)$$

This linear combination of *idf* values underlines the issue that the collection c is much larger than the set r of relevant documents ($|c| >> |r|$), and therefore the maximum-likelihood estimates may be not comparable, and may need to be normalised [13].

Ignoring the negated term events (dropping the term absence, see section 1, variations of BIR term weight) leads to the following simplified term weight:

$$\log \frac{P(t|r)}{P(t|\bar{r})} = idf(t, \bar{r}) - idf(t, r) \approx idf(t, c) - idf(t, r) \quad (26)$$

The formulation shows that the term weight is $idf(t, c)$, if $idf(t, r) = 0$. The latter is the case for terms that occur in *all* relevant documents, i.e., $P(t|r) = 1$. Therefore, from this simplified form of the BIR model, *idf*-based retrieval assumes $P(t|r) = 1$. On the other hand, the full formulation (25) assumes $idf(t, r) = idf(\bar{t}, r)$, i.e., $P(t|r) = 0.5$, for missing relevance.

Regarding BM25, the BIR term weight is in BM25 what IDF is in TF-IDF, i.e., in BM25, a TF-component is multiplied with the BIR F4 term weight, whereas in basic TF-IDF, a TF-component is multiplied with the IDF term weight.

Key Applications

The BIR model is applied to incorporate relevance feedback data into document ranking; this model has become a key foundation of retrieval models. In 2004, the BIR model served as a foundation for a probabilistic ranking of tuples for database queries [1].

Cross-references

- ▶ BM25
- ▶ Language Models
- ▶ TF*IDF

Recommended Reading

1. Chaudhuri S., Das G., Hristidis V., and Weikum G. Probabilistic ranking of database query results. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 888–899.
2. Croft W.B. and Harper D.J. Using probabilistic models of document retrieval without relevance information. J. Doc., 35:285–295, 1979.
3. Grossman D.A. and Frieder O. Information Retrieval. Algorithms and Heuristics, 2nd edn., volume 15 of The Information Retrieval Series. Springer, Berlin, 2004.
4. Harper D.J. and van Rijsbergen C.J. An evaluation of feedback in document retrieval using cooccurrence data. J. Doc., 34:189–216, 1978.
5. Richard K. Belew. Finding out about. Cambridge University Press, 2000.
6. van Rijsbergen C.J. Inform. Retr.. Butterworths, London, 2nd edn., 1979. http://www.dcs.glasgow.ac.uk/Keith/Preface.html.
7. Robertson S. On event spaces and probabilistic models in information retrieval. Inform. Retr. J., 8(2):319–329, 2005.
8. Robertson S.E. The probability ranking principle in IR. J. Doc., 33:294–304, 1977.
9. Robertson S.E. Understanding inverse document frequency: On theoretical arguments for idf. J. Doc., 60:503–520, 2004.
10. Robertson S.E. and Sparck Jones K. Relevance weighting of search terms. J. Am. Soc. Inform. Sci., 27:129–146, 1976.
11. Robertson S.E. and Walker S. On relevance weights with little relevance information. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 16–24.
12. Roelleke T. and Wang J. A parallel derivation of probabilistic information retrieval models. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 107–114.
13. de Vries A. and Roelleke T. Relevance information: a loss of entropy but a gain for IDF? In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 282–289.

Probabilistic Spatial Queries

Reynold Cheng, Jinchuan Chen
The University of Hong Kong, Hong Kong, China

Synonyms

Imprecise spatial queries

Definition

An uncertain item is defined as a range-limited probability density function (pdf) in a multi-dimensional space, which can model the uncertainty of location, sensor and biological data. Given a set of uncertain

items, a probabilistic spatial query returns results augmented with probabilistic guarantees for the validity of answers. The impreciseness of query answers is an inherent property of these applications due to data uncertainty, unlike the techniques for approximate processing that trade accuracy for performance. New query definitions, processing and indexing techniques are required to handle these queries.

Historical Background

Data uncertainty is an inherent property in a number of important and emerging applications. Consider, for example, a habitat monitoring system used in scientific applications, where data such as temperature, humidity, and wind speed are acquired from a sensor network. Due to physical imperfection of the sensor hardware, the data obtained are often inaccurate [9]. Moreover, a sensor cannot report its value at every point in time, and so the system can only obtain data samples at discrete time instants. As another example, in the Global-Positioning System (GPS), the location collected from the GPS-enabled devices (e.g., PDAs) also has measurement and sampling error [16,14]. The location data transmitted to the system may further encounter some network delay. In biometric databases, the attribute values of the feature vectors stored are not exact [1]. Hence, the data collected in these applications are often imprecise, inaccurate, and stale.

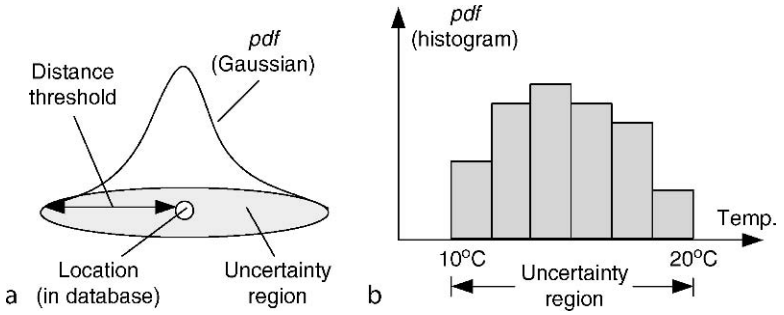
Services or queries that base their decisions on these data can produce erroneous results. There is thus a need to manage these data errors more carefully. In particular, the idea of *probabilistic spatial queries* (PSQ in short), which is a variant of spatial queries that handle data uncertainty, has been recently proposed. The main idea of a PSQ is to consider the models of the data uncertainty (instead of just the data value reported), and augment probabilistic guarantees to the query results. For example, a traditional query asking who is the nearest neighbor of a given point q can tell the user that John is the answer, while a PSQ informs the user that John has a probability of 0.8 of being the closest to q. The probabilities reflect the degree of correctness of query results, thereby facilitating the system to produce a more confident decision.

In this entry, the recent research efforts on PSQ will be summarized. Specifically, the details of how a PSQ can be classified will be discussed. Then, the issues of evaluating and indexing different types of PSQ in a large database will be addressed.

Foundations

Spatial Uncertainty Models To understand a PSQ, it is important to first discuss a commonly-used model of data uncertainty. This model assumes that the actual data value is located within a closed region, called the *uncertainty region*. In this region, a non-zero probability density function (*pdf*) of the value is defined, where the integration of pdf inside the region is equal to one. The cumulative density function (*cdf*) of the item is also provided. In an LBS, a normalized Gaussian pdf is used to model the measurement error of a location stored in a database [16,14] (Fig. 1a). The uncertainty region is a circular area, with a radius called the "distance threshold"; the newest location is reported to the system when it deviates from the old one by more than this threshold (Fig. 1a). Gaussian distributions are also used to model values of a feature vector in biometric databases [1]. Figure 1b shows the histogram of temperature values in a geographical area observed in a week. The pdf, represented as a histogram, is an arbitrary distribution between 10 and 20°C.

A logical formulation of queries for this kind of uncertainty model has been recently studied in [12,15].



Probabilistic Spatial Queries. Figure 1. Location and sensor uncertainty.

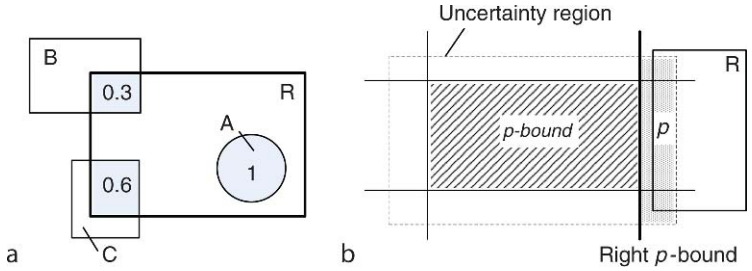
Other variants have also been proposed. In [11], piecewise linear functions are used to approximate the cdf of an uncertain item. Sometimes, point samples are derived from an item's pdf [10,13]. In the *existential uncertainty model*, every object is represented by the value in the space, as well as the probability that this object exists [8]. With these modified models, it is possible to develop fast processing techniques for PSQs.

Query Classification Given the spatial uncertainty model, the semantics of PSQs can be defined. Cheng et al. proposed a classification scheme for different types of PSQ [4]. In that scheme, a PSQ is classified according to the forms of answers. An *entity-based query* is one that returns a set of objects (e.g., list of objects that satisfy a range query or join conditions), whereas a *value-based query* returns a single numeric value (e.g., value of a particular sensor). Another criterion is based on whether an *aggregate* operator is used to produce results. An aggregate query is one where there is interplay between objects that determines the results (e.g., a nearest-neighbor query). Based on these two criteria, four different types of probabilistic queries are defined. Each query type has its own methods for computing answer probabilities. In [4], the notion of *quality* has also been defined for each query type, which provides a metric for measuring the ambiguity of an answer to the PSQ.

In the rest of this section, two important PSQs, namely the probabilistic range queries and the probabilistic nearest-neighbor queries, will be studied.

Probabilistic Range Queries A well-studied PSQ is the *probabilistic range query* (PRQ). Figure 2a illustrates this query, which shows the shape of the uncertainty regions, as well as the user-specified range, R. The task of the range query is to return the each object that can be inside R, as well as its probability. The PRQ can be used in location-based services, where queries like: "return the suspect vehicles in a crime scene" can be asked. It is also used in sensor network monitoring, where sensor IDs whose physical values (e.g., temperature, humidity) are returned to the user. Figure 2a shows the probability values of the items (A, B, and C) that are located inside R. A PRQ is an *entity-based* query, since it returns a list of objects. It is also a *non-aggregate* query, because the probability of each object is independent of the existence of other objects [4].

To compute an item's probability for satisfying the PRQ, one can first find out the overlapping area of each item's region within R (shaded in Fig. 2a), and perform an integration of the item's pdf inside the overlapping area. Unfortunately, this solution may not be very efficient, since expensive numerical integration may need to be performed if the the item's pdf is arbitrary [7]. Even if an R-tree is used to prune items that do not overlap R, the probability of each item that are non-zero still needs to computed. A more efficient solution was developed in [7], where the authors proposed a user-defined constraint, called the *probability threshold* P, with $P \in (0,1]$. An item is only returned if its probability of satisfying the PRQ is not less than P. In Fig. 2a, if $P = 0.6$, then only A and C will be returned to the user. Under this new requirement, it is possible to incorporate the uncertainty information of items into a spatial index (such as R-tree). The main idea is to precompute the *p-bounds* of an item. A p-bound of an uncertain item is essentially a function of p, where $p \in [0,0.5]$. In a 2D space, it is composed of four line segments, as illustrated by the hatched region in Fig. 2b. The requirement of right p-bound (illustrated by the thick solid line) is that the probability of the location of the item on the right of the line has to be exactly equal to p (the shaded area). Similarly, the probability of the item on the left of the left p-bound



Probabilistic Spatial Queries. Figure 2. Probabilistic range queries over uncertain items, showing (a) the probability of each item, and (b) the p-bound of an uncertain item.

is exactly equal to p. The remaining line segments (top and bottom p-bounds) are defined analogously. Once these p-bounds are known, it is possible to know immediately whether an item satisfies the PRQ. Figure 2b shows that a range query R overlaps the item's uncertainty region, but does not cut the right p-bound, where p is less than P. Since the integration of the item's pdf inside the overlapping area of R and the uncertainty region cannot be larger than P, the item is pruned without doing the actual probability computation.

By precomputing a finite number of p-bounds, it is possible to store them in a modified version of the R-tree. Called *Probability Threshold Index* (PTI), this index can facilitate the pruning of uncertain items in the index level [7]. Compared with the R-tree which uses the MBR of the object for pruning, the use of p-bounds in the PTI provides more pruning power. In Fig. 2b, for example, although the range R overlaps with the MBR of the object (dashed rectangle), it does not cut the p-bounds, and so it can be pruned by the PTI but not by the R-tree. [7] also examined special cases of pdf (uniform and Gaussian distributions) and proposed an indexing scheme where p-bounds can be computed on-the-fly without being stored in the index. Since storing p-bounds for high-dimensional uncertain items can be expensive, Tao et al. [17,18] proposed a variant of PTI called *U-tree*, which only stores approximate information of p-bounds in the index. With these improvements, it is possible to index uncertain items in the high-dimensional space. The p-bound techniques were also used in [6] to facilitate the processing of join queries over uncertain spatial data.

Another PRQ evaluation technique was recently proposed by Ljosa et al. [7], who used piecewise linear functions to approximate the cdf of an uncertain item in order to avoid expensive integration. They also described an index that stored these piecewise linear functions, so that a PRQ can be evaluated more efficiently. More recently, the problem of evaluating *imprecise location dependent range queries* is studied [18,2]. This is a variant of PRQ, where the range query is defined with reference to the (imprecise) position of the query issuer. For instance, if the query issuer looks for his friends within 2 miles of his current position, and his position is uncertain, then the actual query range (a circle with a 2-mile radius) cannot be known precisely. The authors of [2] proposed several approaches to efficiently evaluate these queries, by (i) using the Minkowski Sum (a computational geometry technique), (ii) switching the role of query issuer and data being queried, and (iii) using p-bounds.

Nearest-Neighbor Queries Another important PSQ for uncertain items is the probabilistic nearest-neighbor queries (PNNQ in short). This query returns the non-zero probability of each object for being the nearest neighbor of a given point q [4]. A PNNQ can be used in a sensor network, where sensors collect the temperature values in a natural habitat. For data analysis and clustering purposes, a PNNQ can find out the district(s) whose temperature values is (are) the closest to a given centroid. Another example is to find the IDs of sensor(s) that yield the minimum or maximum wind-speed from a given set of sensors [9,4]. A minimum (maximum) query is essentially a special case of PNNQ, since it can be characterized as a PNNQ by setting q to a value of $-\infty$ (∞).

Evaluating a PNNQ is not trivial. In particular, since the exact value of a data item is not known, one needs to consider the item's possible values in its uncertainty region. Moreover, since the PNNQ is an entity-based aggregate query [4], an item's probability depends not just on its own value, but also on the relative values of other objects. If the uncertainty regions of the objects overlap, then their pdfs must be considered in order to derive their corresponding probabilities. This is unlike the evaluation of PRQ, where each item's probability can be computed independent of others. To evaluate PNNQ, one method is to derive the pdf and cdf of each item's distance from q. The probability of an item for satisfying the PNNQ is then computed by integrating over a function of distance pdfs and cdfs [9,4,5]. In [5], an R-tree-based solution for PNNQ was presented. The main idea is to prune items with zero probabilities, using the fact that these items' uncertainty regions must not overlap with that of an item whose maximum distance from q is the minimum in the database. The *probabilistic verifiers*, recently proposed in [3], are algorithms for efficiently computing the lower and upper bounds of each object's probability for satisfying a PNNQ. These algorithms, when used together with the probability threshold defined by the user, avoid the exact probability values to be calculated. In this way, a PNNQ can be evaluated more efficiently.

There are two other solutions for PNNQ that base on a different representation of uncertain items. Kriegel et al. [10] used the Monte-Carlo method, where the pdf of each object was sampled as a set of points. The probability was evaluated by considering the portion of points that could be the nearest neighbor. In [11], Ljosa et al. used piecewise linear representation of the cdf for an uncertain item to propose efficient evaluation and indexing techniques.

Another important entity-based aggregate query over uncertain items, namely the probabilistic skyline queries, has been studied in [13]. A skyline query returns a set of items that are not dominated by other items in all dimensions. In that paper, the issues of defining and computing the probability that an uncertain item was in the skyline were addressed. Two bounding-pruning-refining based algorithms were developed: the bottom-up algorithm used selected instances of uncertain items to prune other instances of uncertain items, while the top-down algorithm recursively partitions the instances of uncertain items into subsets. The authors showed that both techniques enable probabilistic skyline queries to be efficiently computed.

Key Applications

A PSQ can be used in applications that require the processing of uncertain spatial data. These applications include location-based services, road traffic monitoring, wireless sensor network applications, and biometric feature matching, where the data collected from the physical environments (e.g., location, temperature, humidity, images) cannot be obtained with a full accuracy. Recent works that propose to inject uncertainty to a user's location for location privacy protection also requires the use of a PSQ [2].

Future Directions

A lot of work remains to be done in the area of uncertain spatial data processing. An important future work will be the definition and evaluation of important spatial queries, such as reverse nearest-neighbor queries. It will also be interesting to study the development of data mining algorithms for spatial uncertainty. Another direction is to study spatio-temporal queries over historical spatial data (e.g., trajectories of moving objects). Other works include revisiting query cost estimation, query plan evaluation, and user interface design that allows users to visualize uncertain data. A long term goal is to consolidate these research ideas and develop a comprehensive spatio-temporal database system with uncertainty management facilities.

URL to Code

The following URL contains source codes of the ORION system, which is a database system that provides querying facilities for uncertain spatial data: http://orion.cs.purdue.edu

Cross-references

- ▶ Nearest Neighbor Query
- ▶ R-Tree (and Family)
- ▶ Spatial Anonymity
- ▶ Spatial Indexing Techniques

Recommended Reading

1. Böhm C., Pryakhin A., and Schubert M. The Gauss-Tree: Efficient object identification in databases of probabilistic feature vectors. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
2. Chen J. and Cheng R. Efficient evaluation of imprecise location-dependent queries. In Proc. 23rd Int. Conf. on Data Engineering, 2007.
3. Cheng R., Chen J., Mokbel M., and Chow C. Probabilistic verifiers: Evaluating constrained nearest-neighbor queries over uncertain data. In Proc. 24th Int. Conf. on Data Engineering, 2008.
4. Cheng R., Kalashnikov D., and Prabhakar S. Evaluating probabilistic queries over imprecise data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 551–562.
5. Cheng R., Kalashnikov D.V., and Prabhakar S. Querying imprecise data in moving object environments. IEEE Trans. Knowl. and Data Eng., 16(9), 2004.
6. Cheng R., Singh S., Prabhakar S., Shah R., Vitter J., and Xia Y. Efficient join processing over uncertain data. In Proc. Int. Conf. on Information and Knowledge Management, 2006.
7. Cheng R., Xia Y., Prabhakar S., Shah R., and Vitter J. S. Efficient indexing methods for probabilistic threshold queries over uncertain data. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 876–887.
8. Dai X., Yiu M. L., Mamoulis N., Tao Y., and Vaitis M. Probabilistic spatial queries on existentially uncertain data. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 400–417.
9. Deshpande A., Guestrin C., Madden S., Hellerstein J., and Hong W. Model-driven data acquisition in sensor networks. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004.
10. Kriegel H., Kunath P., and Renz M. Probabilistic nearest-neighbor query on uncertain objects. In Proc. 12th Int. Conf. on Database Systems for Advanced Applications, 2007, pp. 337–348.
11. Ljosa V. and Singh A. APLA: Indexing arbitrary probability distributions. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 946–955.
12. Parker A., Subrahmanian V., and Grant J. A logical formulation of probabilistic spatial databases. IEEE Trans. Knowl. and Data Eng., 19(11), 2007.

13. Pei J., Jiang B., Lin X., and Yuan Y. Probabilistic skylines on uncertain data. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007.
14. Pfoser D. and Jensen C. Capturing the uncertainty of moving-objects representations. In Proc. 11th Int. Conf. on Scientific and Statistical Database Management, 1999.
15. Singh S., Mayfield C., Shah R., Prabhakar S., Hambrusch S., Neville J., and Cheng R. Database support for probabilistic attributes and tuples. In Proc. 24th Int. Conf. on Data Engineering, 2008.
16. Sistla P.A., Wolfson O., Chamberlain S., and Dao S. Querying the uncertain position of moving objects. In Temporal Databases: Research and Practice. Springer Verlag, 1998.
17. Tao Y., Cheng R., Xiao X., Ngai W. K., Kao B., and Prabhakar S. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 922–933.
18. Tao Y., Xiao X., and Cheng R. Range Search on Multidimensional Uncertain Data. ACM Trans. Database Syst. 32(3), 2007.

Probabilistic Temporal Databases

V. S. Subrahmanian
University of Maryland, College Park, MD, USA

Synonyms

Temporally uncertain databases; Temporally indeterminate databases

Definition

There are many applications where the fact that a given event occurred is known, but where there is uncertainty about exactly when that event occurred. Such events are called temporally indeterminate events. Probabilistic temporal databases attempt to store information about events that are both temporally determinate and temporally indeterminate. For the latter, they specify a set of time points (often an interval) – it is known that the event occurred at some time point in this set. The probability that the event occurred at a specific time point is given by a probability distribution or by one of a set of probability distributions.

Historical Background

There is no shortage of events that are known to have certainly occurred, but where the exact dates are not known with certainty. For instance, the exact date of the extinction of dinosaurs is unknown – nor does the historical record show the precise date when Cyrus the Great of Persia was born. The latter, estimated by scholars to be between 590–576 BC, is an excellent example of a temporally indeterminate event.

Techniques to study such temporally indeterminate events in databases started with information about *partial null values* [9,13]. Meanwhile, in a largely separate community, researchers focused on the problem of incorporating probabilistic information within a relational database. One of the earliest efforts was due to Cavallo and Pittarelli [4] who proposed an extension of the relational data model to include probabilities – they proposed a partial algebra consisting of projection and join operators. Much early work in this field, such as that of [2,7] made important advances under some restrictions Lakshmanan et al.'s ProbView system [11] proposed two representations of probabilistic data: a *probabilistic tuple* has the form $((t_1,V_1),...,(t_n,V_n))$ where each t_i is a set of possible values and V_i is a probability distribution over the set. They then showed that each probabilistic tuple could be "flattened" into an "annotated" representation. Annotated tuples look like ordinary tuples except that a probability is attached to the tuple as a whole (and a special "path" field was introduced to handle tuples with identical data). ProbView introduced the important concept of conjunction and disjunction strategies that allowed users to specify – in their query – their knowledge about the dependencies between events when posing the query. Thus, the barrier of the independence assumption in previous works was overcome. They proposed a query algebra for annotated tuples, developed query rewrite rules, and developed view maintenance algorithms.

Much work has subsequently been done on temporal probabilistic data. Despite a long history of reasoning about time and uncertainty in AI [10], a major advance in temporal probabilistic databases occurred when Dyreson and Snodgrass [8] proposed the notion of an indeterminate instant. This is just like one of the (t_i,V_i) pairs in the annotated representation of [11] with the exception that the t_i component represents a set of time points. They then extended the SQL query language in several ways. First, they added constructs to indicate that a temporal attribute in indeterminate. Second, they added the concept of "correlation credibility" which allows a query to modify indeterminate temporal data (e.g., by choosing a max temporal value or an expected temporal value). Third, they introduced the concept of "ordering plausibility" which specifies an ordering about the plausibility levels of different conditions in the WHERE clause of an SQL

query. The next major breakthrough in temporal probabilistic databases came when Dekhtyar et al. [6] proposed a temporal probabilistic relational database model that added "tp-cases" to ordinary relational tuples. A tp-case contained two constraints one of which was used to denote valid time points as solutions of the constraints, and the other was used in conjunction with a distribution function to infer a probability distribution on the solutions of the first constraint. They developed an extension of the relational algebra that directly manipulated tp-tuples and used the conjunction and disjunction strategies of ProbView [11] to avoid making independence assumptions. TP-databases were one of the first temporal probabilistic DBMSs to report real world applications for the US Navy [12]. It was later used to build similar applications for the US Army as well. Later, Biazzo et al. [3] extended this work to the case of object bases containing temporal indeterminacy. The Trio system [1] extends temporal indeterminacy models to include lineage information as well. More recently, SPOT databases [14] allow reasoning in the presence of space, time and uncertainty.

Foundations

Consider a relation $R(A_1,...,A_n)$ which describes events whose temporal validity is not precisely known. In order to identify when the tuples in such a relation are valid, *constraints* can be to describe the period of validity of the tuple. The table below (ignoring the shaded column for now) is a temporally indeterminate database about vehicle locations.

The first row in this table says that the event "Vehicle V1 was at location *a*" occurred at some time during the closed interval [11,20]. This is a form of temporal indeterminacy with no probabilities. Suppose there is a probability distribution over the last column. A common temptation is to add an additional column specifying the "name" of the distribution (e.g., "u" might be the uniform distribution, g_r may be a geometric distribution with a parameter r, and so forth). Such a table might now look like this:

VehicleID	Vehicle Type	Location	Time	PDF
V1	T72	*a*	11 < = t < = 20	U
V1	T72	*b*	18 < = t < = 25	$\gamma_{0.5}$
V1	T72	*c*	24 < = t < = 27	$\gamma_{0.2}$

Now consider the table above with the shaded column included. The first row in the table now says that there is a 10% probability that vehicle V1 will be at location *a* at time 11 (and the same for times 12, 13, and so on till time 20) and that the probability is uniformly distributed (probability distribution function or pdf "u"). In contrast, the second row says something different because the pdf γ treats time intervals differently. It says, implicitly, that the probability that vehicle V1 will be at location *b* at time 18 is 0.15, at time 19 is 0.25, at time 20 is 0.125, and so on.

How should queries over this representation of the database be answered? To see this, consider a temporal query which says "Select all tuples where $20 < Time < 23$." The only tuple that has any chance of satisfying this constraint is the second one. However, there is no *guarantee* that the second tuple actually is valid during this interval. Fortunately, there is a 9.375% chance that this is the case – so the system could return the probabilistically valid response saying that the second tuple is valid with probability 9.375%. Unfortunately, there is no way of returning this answer to the user *unless the implicit assumption in all relational databases that the output schema for selection queries should match the input schema is sacrificed*. It is clearly inappropriate to just return the table:

VehicleID	Vehicle Type	Location	Time	PDF
V1	T72	*b*	20 <t<23	$\gamma_{0.5}$

The *Time* field here is computed merely by solving the constraint in the second tuple of the input relation in conjunction with the constraint in the query. Unfortunately, the distribution in the *PDF* field is incorrect, and would need to be recomputed. This is further complicated by the fact that the shape of the original geometric distribution does not look the same when restricted to a portion (of interest) of the original distribution.

There have been two attempts to solve the problem of dealing with what happens when a distribution is manipulated. Dyreson and Snodgrass [8] develop a "rod and point" method to store distributions and infer new distributions when selection operations of the kind above are performed. They approximate a probability mass by splitting it into chunks called "rods." However, each chunk can have a different

length. An approximate representation of the original distribution is obtained through this rod mechanism.

Dekhtyar et al. [6] solve this problem by using some extra space. They require that the "base relation" contains *two constraints*, both of which are identical initially. They would represent the original relation as follows:

VehicleID	Vehicle Type	Location	Time	Time 2	PDF
V1	T72	*a*	11 < = t < = 20	11 < = t < = 20	U
V1	T72	*b*	18 < = t < = 25	18 < = t < = 25	$G_{0.5}$
V1	T72	*c*	24 < = t < = 27	24 < = t < = 27	$G_{0.2}$

In base relations, the *Time* and *Time2* fields have exactly the same constraints in them. When queries are executed, the *Time* field ends up denoting valid time and changes based on the query – however, the *Time2* field rarely changes. When the selection query mentioned above is execution, they would return the answer:

VehicleID	Vehicle Type	Location	Time	Time 2	PDF
V1	T72	*B*	20 < t < 23	18 < = t < = 25	$G_{0.5}$

This is very subtle. The *Time* attribute here specifies the valid time (in this case, time points 21 and 22). The *Time2* attribute is a system-maintained attribute that need not be shown to the user which says apply the PDF mentioned in the *PDF* field to the solutions of the *Time2* constraint – but only show the probability values for the solutions of the *Time* constraint). Thus, *Time2* is used to derive the probabilities for each valid time point. In the above case, the valid time points are 21 and 22, and their probabilities are derived – using the distribution in the *PDF* column applied to the constraint in the *Time2* column – to get probabilities of 0.0625 and 0.03125, respectively. Dekhtyar et al. [6] goes on and specifies how to add additional "low" and "high" probability fields to such relations and provides normalization methods.

Cartesian products between two relations are more complex. When concatenating tuple *st1* and *t2* from two different relations, it is important to consider the probability that both tuples will be valid at a given time point. This requires knowing the relationship between the events being denoted by these two tuples: are they independent? Are they correlated somehow? Is there no information about the relationship? Thus, a *conjunction strategy* must be specified in the query when a Cartesian product (and hence a join operation) is being performed. Dekhtyar et al. [6] show how Cartesian products and joins can be computed under any assumption specified in a user query.

Another recent effort is the one on Trio [1] which attempts to deal with time, uncertainty, and lineage. They address the fact that in many applications involving uncertainty (such as crime-fighting applications), any "final" answer needs to be explainable. As a consequence, they introduce "lineage" parameters when answering a query – informally speaking, the lineage parameter associated with a tuple in an answer (similar to the "path" parameter in [11]) associates a "justification" for each tuple. This justification references the set of base tuples that caused the derived tuple to be placed in an answer.

Key Applications

Temporal probabilistic databases have already been used in defense applications. For instance, [12] describes work in which temporal probabilistic databases are used to store the results of predictions about where enemy submarines will be in the future, when they will be there, and with what probability. In fact, this raises a large set of possible applications based on reasoning about moving objects. For defense applications, cell phone applications, logistics applications, and many other application domains, there is interest in knowing where a moving object will be in the future, when it is expected to be there, and with what probability. Cell phone companies can use such data to understand and better handle load on cell towers.

Moreover, as the world is becoming increasing "geo-location aware" through the use of devices like RFID tags and GPS locators, it is clear that reasoning about where vehicles will be in the future and with what probability will be important in a wide range of applications such as traffic light settings to ease road congestion, recommending detours on highway signs, and more effectively directing 911 traffic in congested situations. These would not be possible without a good estimate of when and where and with what probabilities vehicles will be in the future.

Logistics applications are another important class of applications where companies need to plan activities in the presence of uncertainty about when various supply items will arrive. Corporations today use complex prediction models to learn about suppliers' performance.

Financial applications are another major source of temporal uncertainty. Banks need to have a good idea of their incoming funds. For instance, a credit card provider deals with constant uncertainty about when people will pay their credit card bills, how much of the bills they will pay, and how much they will carry forward as debt. Such applications embody a mix of data uncertainty and temporal uncertainty.

Future Directions

Three major areas of expansion include:

1. *Temporal probabilistic aggregates.* To date, there are almost no techniques to manage aggregates efficiently in temporal probabilistic databases. Most methods would compute aggregates by first answering a non-aggregate query and then deriving aggregates from there: however, techniques to scalably answer aggregate queries are required.
2. *Probabilistic spatio-temporal reasoning.* Moving objects clearly have a spatial component – hence, reasoning about them involves a neat fusion of temporal reasoning, spatial reasoning, and probabilistic reasoning. Some work on indexing in such domains has recently been proposed. However, much future work is needed, especially in understanding the correlations that exist between the presence of a vehicle at time t and its presence at another location at time $t + 1$.
3. *Query optimization.* Recent work [5] has made a good start on query optimization in probabilistic databases – however, query optimization in databases involving time and uncertainty has a ways to go. Such methods are critical for scaling applications.

Cross-references

► Qualitative Temporal Reasoning
► Temporal Constraints

Recommended Reading

1. Agrawal P., Benjelloun O., Sarma A.D., Hayworth C., Nabar S.U., Sugihara T., and Widom J. Trio: a system for data, uncertainty, and Lineage. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 1151–1154.
2. Barbará D., Garcia-Molina H., and Porter D. The management of probabilistic data. IEEE Trans. Knowl. Data Eng., 4(5): 487–502, 1992.
3. Biazzo V., Giugno R., Lukasiewicz T., and Subrahmanian V.S. Temporal probabilistic object bases. IEEE Trans. Knowl. Data Eng., 15(4):921–939, 2003.
4. Cavallo R. and Pittarelli M. The theory of probabilistic databases. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 71–81.
5. Dalvi N. and Suciu D. Answering queries from statistics and probabilistic views. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 805–816.
6. Dekhtyar A., Ross R., and Subrahmanian V.S. Probabilistic temporal databases, I: algebra. ACM Trans. Database Syst., 26(1): 41–95, 2001.
7. Dey D. and Sarkar S. A probabilistic relational model and algebra. ACM Trans. Database Syst., 21(3):339–369, 1996.
8. Dyreson C.E. and Snodgrass R.T. Supporting valid-time indeterminacy. ACM Trans. Database Syst., 23(1):1–57, 1998.
9. Grant J. Partial values in a tabular database model. Inf. Process. Lett., 9(2):97–99, 1979.
10. Kraus S. and Subrahmanian V.S. Multiagent reasoning with probability, time and beliefs. Int. J. Intell. Syst., 10(5):459–499, 1994.
11. Lakshmanan L.V.S., Leone N., Ross R.B., and Subrahmanian V.S. ProbView: a flexible probabilistic database system. ACM Trans. Database Syst., 22(3):419–469, 1997.
12. Mittu R. and Ross R. Building upon the coalitions agent experiment (COAX) – integration of multimedia information in GCCS-M using IMPACT. In Proc. Ninth Int. Workshop on Multimedia Information Systems, 2003, pp. 35–44.
13. Ola A. Relational databases with exclusive disjunctions. In Proc. 8th Int. Conf. on Data Engineering, 1992, pp. 328–336.
14. Parker A., Subrahmanian V.S., and Grant J. A logical formulation of probabilistic spatial databases. IEEE Trans. Knowl. Data Eng., 19(11):1541–1556.

Probability Ranking Principle

Ben He
University of Glasgow, Glasgow, UK

Synonyms

PRP

Definition

The probability ranking principle asserts that relevance has a probabilistic interpretation. According to this principle documents are ranked by a probability $p(Rel|d, q)$, where Rel denotes the event of a document d being relevant to a query q. Robertson called this principle the *probability ranking principle* [1].

Key Points

By assuming independence between query terms, Robertson and Sparck-Jones proposed for the probability $p(Rel|d,q)$ the following model (the RSJ model [2]):

$$\log(p(Rel|d,q)) \propto \sum_{t \in q} \log \frac{p(t|Rel) \cdot p(\overline{t}|\overline{Rel})}{p(t|\overline{Rel}) \cdot p(\overline{t}|Rel)} \quad (1)$$

where $\overline{Rel}$ indicates the event of non-relevance; t and $\overline{t}$ indicate the events that the term t occurs in document d or does not, respectively. For each query term t, the probability $p(Rel|d,t)$ is given by the sum of two log-odds, $\log \frac{p(t|Rel)}{p(t|\overline{Rel})}$ and $\log \frac{p(\overline{t}|\overline{Rel})}{p(\overline{t}|Rel)}$.

If N is the number of documents in the whole collection, R is the number of relevant documents, r is the number of relevant documents containing t, N_t is the document frequency, i.e., the number of documents containing t, [3] instantiated the RSJ model as follows:

$$w^{(1)} = \log \frac{(r+0.5)(N - N_t - R + r + 0.5)}{(R - r + 0.5)(N_t - r + 0.5)} \quad (2)$$

where $w^{(1)}$ is the raw weight of a term t in a document d. The number 0.5 is used to avoid assigning negative weights. The formula is called the "point-5" formula.

If relevance information is not available, i.e., $R = r = 0$, the point-5 formula can be written as:

$$w^{(1)} = \log \frac{N - N_t + 0.5}{N_t + 0.5} \quad (3)$$

As one of the most well-established IR systems, Okapi uses a weighting model that is based on the RSJ model introduced above, and takes also term frequency (tf) and query term frequency (qtf) into consideration.

Cross-references

- ▶ Information Retrieval
- ▶ Information Retrieval Models
- ▶ Term weighting

Recommended Reading

1. Robertson S.E. The probability ranking principle in IR. J. Doc., 33:294–304, 1977.
2. Robertson S.E. and Sparck-Jones K. Relevance weighting of search terms. J. Am. Soc. Inf. Sci., 27:129–146, 1977.
3. Robertson S.E. and Walker S. On relevance weights with little relevance information. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 16–24.

Probability Smoothing

Djoerd Hiemstra
University of Twente, AE Enschede, The Netherlands

Definition

Probability smoothing is a language modeling technique that assigns some non-zero probability to events that were unseen in the training data. This has the effect that the probability mass is divided over more events, hence the probability distribution becomes more *smooth*.

Key Points

Smoothing overcomes the so-called *sparse data problem*, that is, many events that are plausible in reality are not found in the data used to estimate probabilities. When using maximum likelihood estimates, unseen events are assigned zero probability. In case of information retrieval, most events are unseen in the data, even if simple unigram language models are used documents are relatively short (say on average several hundreds of words), whereas the vocabulary is typically big (maybe millions of words), so the vast majority of words does not occur in the document. A small document about "information retrieval" might not mention the word "search," but that does not mean it is not relevant to the query "text search." The sparse data problem is the reason that it is hard for information retrieval systems to obtain high recall values without degrading values for precision, and smoothing is a means to increase recall (possibly degrading precision in the process). Many approaches to smoothing are proposed in the field of automatic speech recognition [1]. A smoothing method may be as simple so-called Laplace smoothing, which adds an extra count to every possible word. The following equations show respectively (1) the unsmoothed, or maximum likelihood estimate, (2) Laplace smoothing, (3) Linear interpolation smoothing, and (4) Dirichlet smoothing [3]:

$$P_{ML}(T = t|D = d) = tf(t,d) / \sum_{t'} tf(t',d) \quad (1)$$

$$P_{LP}(T = t|D = d) = (tf(t,d) + 1) / \sum_{t'} (tf(t',d) + 1) \quad (2)$$

$$P_{LI}(T = t|D = d) = \lambda P_{ML}(T = t|D = d) + (1 - \lambda)P_{ML}(T = t|C) \quad (3)$$

$$P_{Di}(T = t|D = d) = (tf(t, d) + \mu P_{ML}(T = t|C)) / ((\sum_{t'} tf(t', d)) + \mu) \quad (4)$$

Here, $tf(t,d)$ is the frequency of occurrence of the term t in the document d, and $P_{ML}(T|C)$ is the probability of a term occurring in the entire collection C. Both linear interpolation smoothing and Dirichlet smoothing assign a probability proportional to the term occurrence in the collection to unseen terms. Here, λ ($0<\lambda<1$) and μ ($\mu>0$) are unknown parameters that should be tuned to optimize retrieval effectiveness. Linear interpolation smoothing has the same effect on all documents, whereas Dirichlet smoothing has a relatively big effect on small documents, but a relatively small effect on bigger documents. Many smoothed estimators used for language models in information retrieval (including Laplace and Dirichlet smoothing) are approximations to the *Bayesian predictive distribution* [2].

Cross-references

► Language Models
► N-Gram Models

Recommended Reading

1. Chen S.F. and Goodman J. An empirical study of smoothing techniques for language modeling. Technical report TR-10-98, Center for Research in Computing Technology, Harvard University, August 1998.
2. Zaragoza H., Hiemstra D., Tipping M., and Robertson S. Bayesian extension to the language model for ad hoc information retrieval. In Proc. 26th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2003, pp. 4–9.
3. Zhai C. and Lafferty J. A study of smoothing methods for language models applied to information retrieval. ACM Trans. Inf. Syst., 22(2):179–214, 2004.

Procedure Order

► Clinical Order

Procedure Request

► Clinical Order

Process Composition

► Composition

Process Definition

► Workflow Model

Process Evolution

► Workflow Evolution

Process Life Cycle

Nathaniel Palmer
Workflow Management Coalition, Hingham, MA, USA

Synonyms

Workflow lifecycle; Thread lifecycle; Process state model

Definition

The stages of life from the start to the end of a process instance within the context of workflow management.

Key Points

The Process Life Cycle represents the stages of a process instance as it evolves from instantiation to termination. This life cycle is most closely related to the life cycle of a thread, and is distinct from the life cycle approach to Business Process Management initiatives, involving an iterative or recursive evolution through the five stages of design, modeling, execution, monitoring, and optimization.

The latter notion of Business Process Management Life cycle is associated with the discipline of continuous process improvement, whereby processes are

never deemed "complete" and thus no longer subject to change, but rather are continuously improved through multiple instances of execution, examination, and modification. In contrast, the Process Life Cycle as defined herein refers to the "life span" of a process instance and has definitive start and end points. Thus the Process Life Cycle of the individual process instance is more aptly described as linear as opposed to cyclical, although at various steps in the process it may cycle between running and suspended states.

The steps of the Process Life Cycle are "Instantiate" representing the creation of a new instance (making it live but not necessarily running); "activate" representing the activation of the process instance (now live and running); "passivate" which refers to temporarily suspending the instance (live but not running); "terminate" which represents the end of life of the process instances, through either abortion, cancellation, or completion after running through the full process as defined.

Cross-references

- ▶ Business Process Model
- ▶ Process Definition
- ▶ Workflow Model

Process Management

- ▶ Business Process Management

Process Mining

W. M. P. van der Aalst
Eindhoven University of Technology, Eindhoven, The Netherlands

Synonyms

Workflow mining

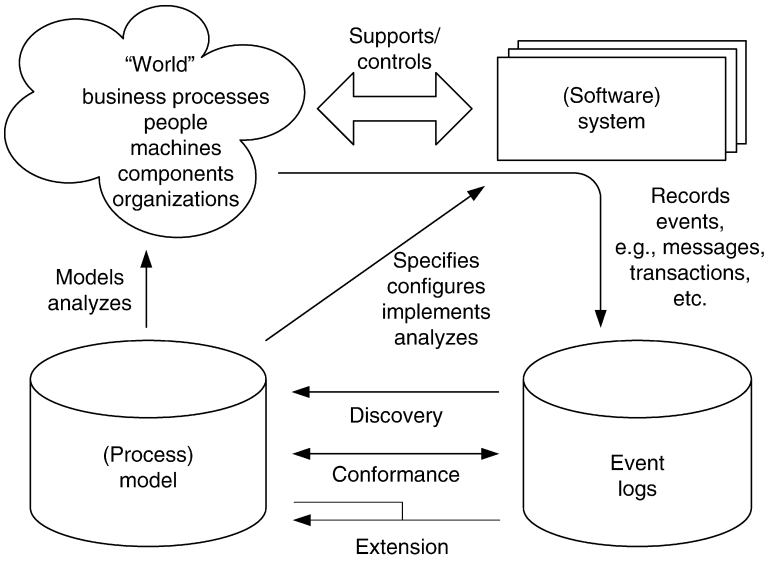
Definition

Process mining techniques allow for the analysis of business processes based on event logs. For example, the audit trails of a workflow management system, the transaction logs of an enterprise resource planning system, and the electronic patient records in a hospital can be used to discover models describing processes, organizations, and products. Moreover, such event logs can also be used to compare event logs with some a priori model to see whether the observed reality conforms to some prescriptive or descriptive model.

The basic idea of *process mining* is to discover, monitor, and improve *real* processes (i.e., not assumed processes) by extracting knowledge from event logs. Today many of the activities occurring in processes are either supported or monitored by information systems. Consider for example ERP, WFM, CRM, SCM, and PDM systems to support a wide variety of business processes while recording well-structured and detailed event logs. However, process mining is not limited to information systems and can also be used to monitor other operational processes or systems. For example, process mining has been applied to complex X-ray machines, high-end copiers, web services, careflows in hospitals, etc. All of these applications have in common that *there is a notion of a process* and that *the occurrences of activities are recorded in so-called event logs*. Assuming that the supporting systems log events, a wide range of *process mining techniques* comes into reach. The basic idea of process mining is to learn from observed executions of a process and can be used to (i) *discover* new models (e.g., constructing a Petri net that is able to reproduce the observed behavior), (ii) check the *conformance* of a model by checking whether the modeled behavior matches the observed behavior, and (iii) *extend* an existing model by projecting information extracted from the logs onto some initial model (e.g., show bottlenecks in a process model by analyzing the event log). All three types of analysis have in common that they assume the existence of some *event log*.

Key Points

The goal of process mining is to discover, monitor, and improve real processes by extracting knowledge from event logs. Clearly, process mining is relevant in a setting where much flexibility is allowed or needed, because the more ways in which people and organizations can deviate, the more variability and the more interesting it is to observe and analyze processes as they are executed. Three basic types of process mining can be identified (Fig. 1):



Process Mining. Figure 1. Three types of process mining: (i) discovery, (ii) conformance, and (ii) extension.

1. *Discovery*. There is no a priori model, i.e., based on an event log some model is constructed. For example, using the α-algorithm [2] a process model can be discovered based on low-level events.
2. *Conformance*. There is an a priori model. This model is used to check if reality conforms to the model. For example, there may be a process model indicating that purchase orders of more than one million Euro require two checks. Another example is the so-called four-eyes principle where two activities need to be executed by different people. Conformance checking may be used to detect deviations, to locate and explain these deviations, and to measure the severity of these deviations.
3. *Extension*. There is an a priori model. This model is extended with a new aspect or perspective, i.e., the goal is not to check conformance but to enrich the model. An example is the extension of a process model with performance data, i.e., some a priori process model dynamically annotated with performance data (e.g., bottlenecks are shown by coloring parts of the process model). Figure 1 shows that a log and a model are used to create a new model.

Traditionally, process mining has been focusing on *discovery*, i.e., deriving information about the original process model, the organizational context, and execution properties from enactment logs. An example of a technique addressing the control flow perspective is the α-algorithm, which constructs a Petri net model describing the behavior observed in the event log. However, process mining is not limited to process models (i.e., control flow) and recent process mining techniques are more and more focusing on other perspectives, e.g., the organizational perspective or the data perspective. For example, there are approaches to extract social networks from event logs and analyze them using social network analysis [1]. This allows organizations to monitor how people and groups are working together.

Conformance checking compares an a priori model with the observed behavior as recorded in the log. In [4] it is shown how a process model (e.g., a Petri net) can be evaluated in the context of a log using metrics such as "fitness" (Is the observed behavior possible according to the model?) and "appropriateness" (Is the model "typical" for the observed behavior?). However, it is also possible to check conformance based on organizational models, predefined business rules, temporal formula's, Quality of Service (QoS) definitions, etc.

There are different ways to *extend* a given process model with additional perspectives based on event logs, e.g., decision mining. Decision mining, also referred to as decision point analysis, aims at the detection of data dependencies that affect the routing of a case. Starting from a process model, one can analyze how data attributes influence the choices

made in the process based on past process executions. Classical data mining techniques such as decision trees can be leveraged for this purpose. Similarly, the process model can be extended with timing information (e.g., bottleneck analysis).

Process mining is strongly related to classical data mining approaches. However, the focus is not on data but on process-related information (e.g., the ordering of activities). Process mining is also related to monitoring and business intelligence [3].

Cross-references

- ▶ Association Rule Mining on Streams
- ▶ Data Mining
- ▶ Workflow Management

Recommended Reading

1. van der Aalst W.M.P. Reijers H.A., and Song M. Discovering social networks from event logs. Comput. Support. Coop. Work, 14(6):549–593, 2005.
2. van der Aalst W.M.P. Weijters A.J.M.M., and Maruster L. Workflow mining: discovering process models from event logs. IEEE Trans. Knowl. Data Eng., 16(9):1128–1142, 2004.
3. Grigori D., Casati F., Castellanos M., Dayal U., Sayal M., and Shan M.C. Business process intelligence. Comput. Ind., 53(3):321–343, 2004.
4. Rozinat A. and van der Aalst W. M. P. Conformance checking of processes based on monitoring real behavior. Inf. Syst., 33(1):64–95, 2007.

Process Optimization

Danilo Ardagna
Politechnico di Milano University, Milan, Italy

Synonyms

Business process optimization; QoS-based web services composition

Definition

With the development of the service oriented architecture (SOA), complex applications can be composed as business processes invoking a variety of available Web services (WSs) with different characteristics. Advanced SOA systems [1,12,15] allow the development of applications by specifying component WSs in a process only through their required functional characteristics, and to select WSs during process execution from the ones included in a registry of available services. Service descriptions are stored and retrieved from enhanced UDDI registries, which also provide information about quality of service (QoS) on the provider side. Usually, a set of functionally equivalent services can be selected, i.e., services which implement the same functionality but differ for the quality parameters.

Process optimization identifies the best set of services available at run time, taking into consideration end-user preferences and constraints on the QoS properties.

Historical Background

Process optimization has its roots in workflow scheduling problems. Scheduling of workflows [13] is the problem of finding a correct execution sequence of workflow tasks such that some temporal constraints or resource constraints (i.e., agents which can support tasks executions) are met. In workflow management systems, agents could be human beings or software applications, in Process optimization the available resources are component WSs. Process optimization is related also to workflow/process planning [3] where the problem of synthesizining a complex behavior from an explicit goal and a set of candidates which contribute to a partial reaching of this goal is investigated. In Process optimization, vice versa, the *process schema*, i.e., the sequence of activities, is given and the optimum mapping of activities to component WSs candidate for their execution is identified.

Process optimization has been applied in context-aware business processes and e-science research fields. The literature has provided *three generations* of solutions. First generation solutions implemented *local* approaches [3,14,15] which select WSs one at the time by associating the running abstract activity to the best candidate service which supports its execution. Local approaches can guarantee only local QoS constraints, i.e., candidate WSs are selected according to a desired characteristic, e.g., the price of *a single WS* is lower than a given threshold.

Second generation solutions proposed *global* approaches [5,8,10,12,15]. The set of services which satisfy the process constraints and user preferences for the whole application are identified before executing the process. In this way, QoS constraints can predicate at a global level, i.e., constraints posing restrictions over the *whole composed service execution*

can be introduced. In order to guarantee the fulfilment of global QoS constraints, second generation optimization techniques consider the worst case execution scenario for the composed service. For cyclic processes, loops are unfolded, i.e., unrolled according to their maximum number of iterations [5,15]. This approach could be very conservative and constitutes the main limitation of second generation techniques. Furthermore, global approaches introduce an increased complexity with respect to local solutions. The main issue for the fulfillment of global constraints is WSs performance variability. Indeed, the QoS of a WS may evolve relatively frequently, either because of internal changes or because of workload fluctuations [15]. If a business process has a long duration, the set of services identified by the optimization may change their QoS properties during the process execution or some services can become unavailable or others may emerge. In order to guarantee global constraints, WS selection and execution are interleaved: optimization is performed when the business process is instantiated and its execution is started, and is iterated during the process execution performing *re-optimization* at run time.

To reduce optimization/re-optimization complexity, a number of solution have been proposed which guarantee global constraints only for the critical path [15] (i.e., the path which corresponds to the highest execution time), or reduce loops to a single task [5], satisfying global constraints only statistically, by applying the reduction formula proposed in [7].

Another drawback of second generation solutions is that, if the end-user introduces severe QoS constraints for the composed service execution, i.e., limited resources which set the problem close to un-feasibility conditions (e.g., limited budget or stringent execution time limit), no solutions could be identified and the composed service execution fails [5].

Third generation techniques [3] overcome the limits of the previous approaches and focus on the execution of processes under severe QoS constraints. Severe constraints are very relevant whenever processes have to be performed with stringently limited resources. Third generation solutions are based on loops peeling, which significantly improves the solutions based on loops unfolding. Furthermore, negotiation is exploited if a feasible solution cannot be identified, to bargain QoS parameters with service providers offering services, reducing process invocation failures.

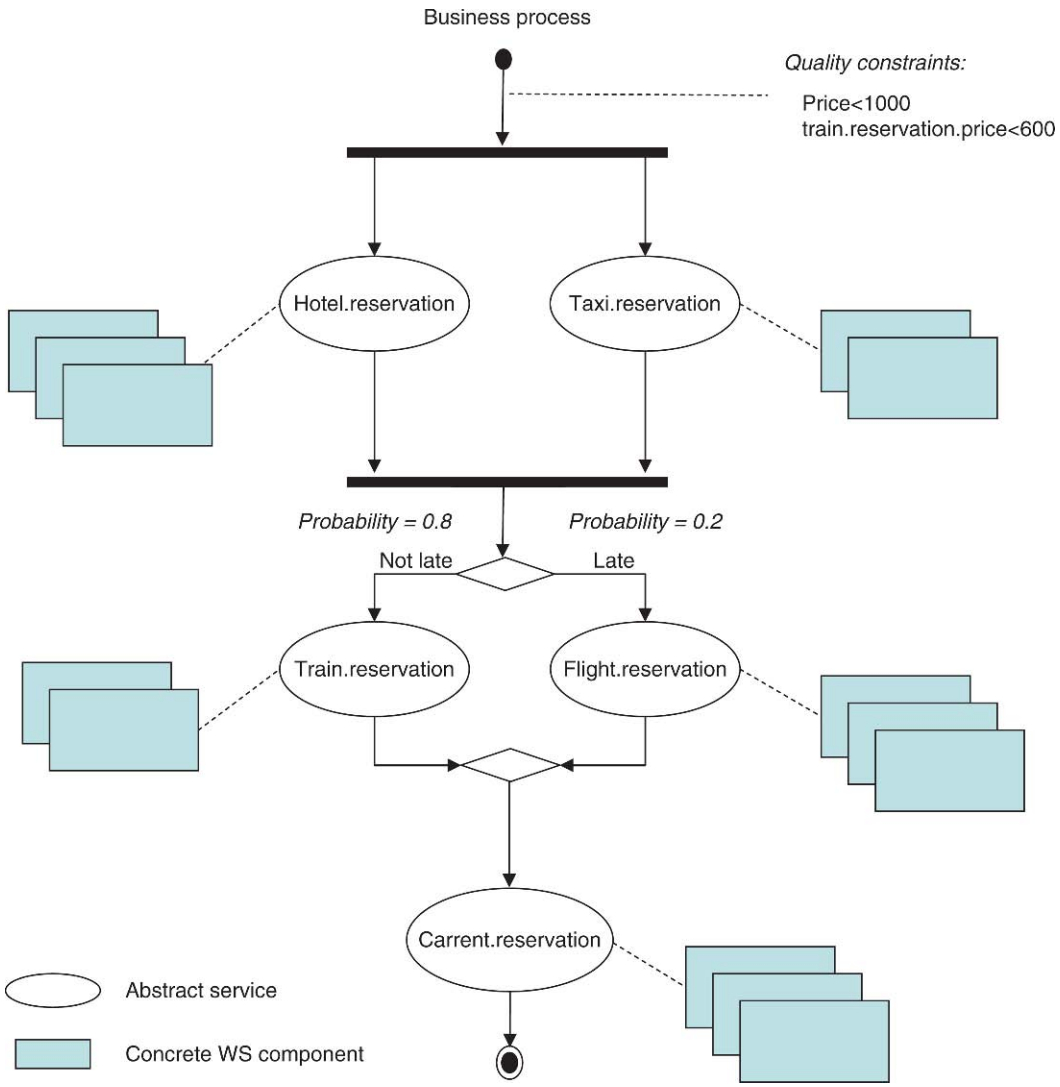
Foundations

Process optimization allows the specification of complex applications as business processes composed by *abstract services* which act as *place holders* of WS components invoked at run time. The best set of services, selected by solving an optimization problem, is invoked at run time by implementing a *dynamic/late binding* mechanism.

Process optimization is usually formalized as a multi-objective optimization problem since several quality criteria can be associated with WS execution. Past approaches [5,12,15] focussed on *execution time* (the expected delay, between the time instant when a request is sent and the time when the result is obtained), *availability* (the probability that the service is accessible), *price* (the fee that a service requester has to pay to the Service Provider for the service invocation), and *reputation* (a measure of the service trustworthiness). Furthermore, the optimization is performed statistically, i.e., by considering the probability of execution of the execution paths of the business process (i.e., any possible sequence of invocations of abstract services). For this reason, some annotations are added to the BPEL specification in order to identify: (i) the maximum [15] or the probability distribution [3] of the number of iterations of loops; (ii) the expected frequency of execution of conditional branches; (iii) global and local constraints on quality dimensions.

Figure 1 shows an example of composed process which implements a virtual travel agency, and the corresponding annotations which specify constraints. The BPEL specification includes invocation to abstract WSs which can be supported at run time by *concrete* WS components.

The probability distribution of the number of iterations of loops and the frequency of execution of conditional branches can be evaluated from past executions by inspecting system logs or can be specified by the composite service designer. If an upper bound for loops execution cannot be determined, then the optimization cannot guarantee that global constraints are satisfied [15]. Prior to perform process optimization, loops are unfolded [15] or peeled [3] (see Fig. 2) and are modeled as directed acyclic graphs (DAGs). Loops peeling is a form of loops unrolling where loop iterations are represented as a sequence of branches and each branch condition evaluates if the loop has to continue with the next ith iteration or it has to exit with probability p_i.

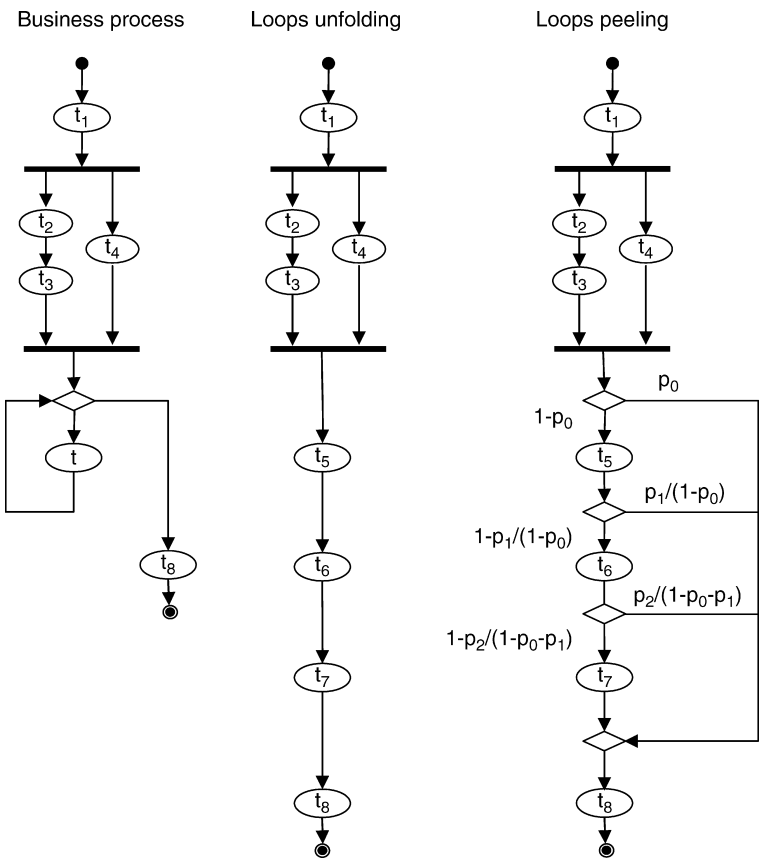


Process Optimization. Figure 1. Virtual travel agency process specification.

The objective function to be optimized is, usually [3,14,15], the aggregated value of QoS for the end user which can be obtained by applying the simple additive weighting (SAW) technique. SAW is one of the most widely used techniques to obtain a *score* from a list of dimensions. Since the quality dimensions have different units of measure, the SAW method first normalizes the raw values for each quality dimension. Each quality dimension is also associated with a weight which expresses the user preferences among multiple quality parameters. The overall value of QoS is calculated as a weighted sum of the normalized values of quality dimension. The SAW method originates a linear objective function, other proposal introduces more general utility functions (i.e., functions which map each possible configuration of the business process to a scalar value) which can be non-linear [5].

First generation solutions considered only *local constraints*. In that case, the process optimization is very simple and the optimum solution can be identified by a greedy algorithm which selects the best candidate service suitable for the execution. An example of first generation technique can be found in [11], where Web agents can migrate to invoke services locally in order to minimize also the network bandwidth.

Second generation solutions support *global constraints* and introuce NP-hard optimization problems. In [4] the complexity of some variants of the global process optimization problem is analyzed, while an overview of heuristic techniques, which hence identify



Process Optimization. Figure 2. Loops unfolding and peeling.

only sub-optimal solutions, can be found in [10]. In [14] global process optimization has been modeled as a multiple choice multiple dimension knapsack problem (MMKP) and as a graph constrained optimum path problem. A MMKP is one kind of knapsack problem where the resources are multidimensional, that is, there are multiple resource constrains for the knapsack (e.g., weight and volume) and items are classied in groups. Each item of the group has a particular value and it requires resources. The objective of the MMKP is to pick exactly one item from each group for maximum total value of the collected items, subject to the resource constraints of the knapsack. Process optimization can be reduced to a MMKP since each abstract service corresponds to a group, each concrete WS is an item in a group and each QoS constraint corresponds dimension of the knapsack. Authors in [14] has implemented ad hoc efficient techniques to identify sub-optimal solutions of the MMKP.

Global approaches have been proposed for the first time in [15], where the Process optimization problem has been formalized as a mixed integer linear programming problem, solved by integer linear programming solvers. The authors separately optimize each execution path and obtain the abstract services to concrete services mapping by composing separate solutions according to the frequency of execution. This approach has some limitations (e.g., availability and response time constraints are guaranteed only for the critical path, and global constraints cannot always be fulfilled) which have been solved by following research proposals [3,14].

Some recent proposals face the Process optimization problem by implementing genetic algorithms [5,8]. In Canfora et al. [5] the reduction formulas presented in [7] are adopted, the re-optimization is considered but abstract services specified in loops are always assigned to the same concrete WS component. Furthermore, by applying reduction formulas the solution guarantees global constraints only statistically. At run time, if low probability paths are taken (see [5]), then the solution could become infeasible and re-optimization must be triggered. In [8], the

multi-objective evolutionary approach NSGA-II (non-dominated sorting genetic algorithm) is implemented, which identifies a set of Pareto optimal solutions without introducing a ranking among different quality dimensions. Every identified solution is characterized by the fact that no other plans exist such that a quality dimension is improved without worsening the other ones. Genetic algorithms are more flexible than mixed integer linear approaches, since they allow considering also non-linear composition rules for composed WSs, but are less computationally efficient. In current implementations, some execution time is wasted by generating also non-feasible solutions and, sometimes, no solution can be identified even when the problem is feasible in case the global constraints are stringent.

The work presented in [3] proposes a third generation solution which poses the basis for the execution of processes under severe QoS constraints. The solution is based on loops peeling which significantly improves the solutions based on loops unfolding (up to 40%). Furtheremore, negotiation techniques are exploited to identify a feasible solution of the problem, if one does not exist, reducing process invocation failures. The joint optimization and negotiation approach has been proved to be effective for large processes (including up to 10,000 abstract services), when QoS constraints are severe and reduces also the re-optimization overhead.

Key Applications

Context-Aware Business Process

Dynamic WS selection for composed WSs focused in particular on context aware business processes. Context awareness may be needed both when considering WS personalization, where a generic process is personalized choosing services according to user preferences, and in mobile composed services, to provide ubiquitous services where selection and execution depend on the available services and their QoS [1].

E-Science and Grid Computing

In e-science complex processes, defined as workflows enacted in grid environments, are being developed reaching the dimension of thousands of tasks in "in silico" experiments [9]. Each task is performed selecting and invoking a service. In this case, the Process optimization problem is more challenging, since the concrete resources have to be modeled with a more fine grain and are represented by the physical machines which can support tasks execution instead of abstract WSs. Current solutions propose to create a hierarchy of processes or distributing the workflow over a number of engines, partitioning in this way the optimization problem but leading to sub-optimal solutions.

Future Directions

All of the above approaches consider the optimization of a single process instance and assume a constant QoS profile. Cardellini et al. [6] tackled the problem of optimization of multiple process instances in order to reduce optimization overhead. The work presented in [2] considered variable (periodic) quality of service profiles of component WSs and explicitly addressed long term process execution. The execution of multiple instances, variable QoS profiles, and long term process execution make the optimization problem more cumbersome. Only heuristic approaches have been proposed so far, more efficient solutions, both in term of optimization time and quality of the final solution, are needed.

Experimental Results

For every presented approach, there is an experimental evaluation in the corresponding reference. Zeng et al. [15] discuss local and global approaches. Tao Yu et al. [14] present a comparison among linear integer programming approaches and heuristic solutions. The work presented in [3] analyzes loops peeling and second and third generation solutions.

Cross-references

▶ Composed Services and WS-BPEL
▶ Grid Workflow
▶ Workflow Management and Workflow Managemenet System

Recommended Reading

1. Ardagna D., Comuzzi M., Mussi E., Pernici B., and Plebani P. PAWS: A Framework for Executing Adaptive Web-Service Processes. IEEE Software, 24(6):39–46, 2007.
2. Ardagna D., Giunta G., Ingraffia N., Mirandola R., and Pernici B. QoS-driven Web services selection in autonomic grid environments. In Proc. OTM Confederated Int. Conf. CODPIS, DOA, GADA, and ODBASE, 2006, pp. 1273–1289.
3. Ardagna D. and Pernici B. Adaptive Service Composition in Flexible Processes. IEEE Trans. Software Eng., 2007.
4. Bonatti P.A. and Festa P. On optimal service selection. In Proc. 14th Int. World Wide Web Conference, 2005, pp. 530–538.
5. Canfora G., Penta M., Esposito R., and Villani M.L. QoS-Aware Replanning of Composite Web Services. In Proc. IEEE Int. Conf. on Web Services, 2005.

6. Cardellini V., Casalicchio E., Grassi V., and Mirandola R. A framework for optimal service selection in broker-based architectures with multiple QoS classes. In Proc. IEEE Services Comput. Workshops, 2006, pp. 105–112.
7. Cardoso J. Quality of Service and Semantic Composition of Workflows, Ph. D. Thesis, Univ. of Georgia, 2002.
8. Claro D.B., Albers P., and Hao J.K. Selecting Web Services for Optimal Composition. In Proc. IEEE Int. Conf. on Web Services., 2005.
9. Fox G.C. and Gannon D. Workflow in Grid Systems. Concurrency and Computation: Practice and Experience, 18(10):1009–1019, 2006.
10. Jaeger M.C., Muhl G., and Golze S. QoS-Aware Composition of Web Services: An Evaluation of Selection Algorithms. In Proc. Int. Conf. on Cooperative Inf. Syst., 2005.
11. Maamar Z., Sheng Q.Z., and Benatallah B. Interleaving Web Services Composition and Execution Using Software Agents and Delegation. In Proc. Web Services and Agent-Based Eng., 2003.
12. Patil A.A., Oundhakar S.A., Sheth A.P., and Verma K. METEOR-S web service annotation framework. In Proc. 12th Int. World Wide Web Conference, 2004, pp. 553–562.
13. Senkul P. and Toroslu I.H. An architecture for workflow scheduling under resource allocation constraints. Inf. Syst., 30(5):399–422, 2005.
14. Yu T., Zhang Y., and Lin K.J. Efficient algorithms for Web services selection with end-to-end QoS constraints. ACM Trans. Web, 1(1):1–26, 2007.
15. Zeng L., Benatallah B., Dumas M., Kalagnamam J., and Chang H. QoS-Aware Middleware for Web Services Composition. IEEE Trans. Software Eng., 30(5), 2004.

Process Semantics

► Workflow Constructs

Process State Model

► Process Life Cycle

Process Structure of a DBMS

Pat Helland
Microsoft Corporation, Redmond, WA, USA

Synonyms

Cluster databases; Scale-out databases; Scale-up databases; Shared-disk databases; Shared-nothing databases; Shared-everything databases

Definition

Database Management Systems are typically implemented on top of operating systems which allow execution within processes. Different systems have chosen different process structures as they map their computation onto the operating system. This section surveys some of these choices.

Historical Background

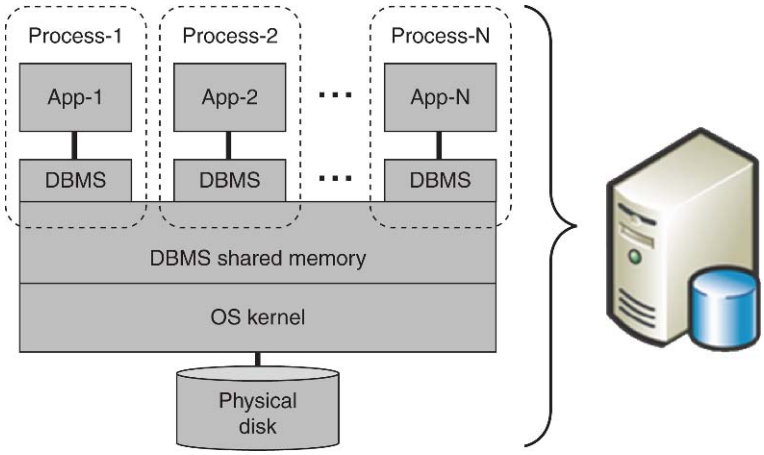
The first database management systems were simple libraries that ran inside the process of the application. While the use of these libraries offered leverage to the applications by providing essential functionality, they did not offer protection for the data in the presence of application errors.

To provide protection, DBMSs were initially moved into higher security rings accessible by hardware protected transitions to memory and code which was more secure than the application but less secure than the operating system kernel. Running the DBMS in shared (but secured) memory allowed access by multiple applications (in separate processes). The shared memory within the DBMS allowed for efficient cross application management of data (See Fig. 1).

Two trends caused gradual retreat from the implementation of the DBMS within a trusted security ring. First, there is the emergence of DBMSs that were designed to be ported across operating systems. Second, the emergence of distributed computing drove the need to run different portions of the computing stack (both application and database) across different machines. Both of these trends grew throughout the 1980s and led to a constellation of process and processor architectures for both applications and database management systems.

The process structure of database management systems has evolved and today can be seen in many forms. As mentioned above, initial implementations of DBMS systems in the 1960s and 1970s were embedded in the same process (but soon with protection for the DBMS within a security ring).

In the 1980s, a number of distributed databases emerged, exemplified by Tandem's NonStop SQL. In these, the application interface remained an in-memory call to portions of the database system but behind the scenes there were cross-process and cross-processor calls to other portions of the DBMS. This was implemented transparently to the application except, of course, with some performance implications which could be both positive and negative.



Process Structure of a DBMS. Figure 1. Early implementations of DBMS systems used shared memory as a technique to allow the DBMS to run in process with the application. This minimized app to DBMS communication costs.

In the 1990s, the client-server computing first arrived with the separation of the client and server-side database in what became known as *two-tier client-server* applications. This necessitated the creation of the *database connection*, exemplified by ODBC (Open Database Connectivity). With a database connection, both DML (Data Manipulation Language) and the resultant data sets were returned across process boundaries, allowing the application process (and processor) to be different than the database process and processor. Subsequently, the application itself began to break across processes and processors resulting in *three-tier client-server* or even later *N-tier client-server systems*. Within each of these architectures, the application program still perceived the notion of a single database even though the work was potentially spread across multiple systems.

Shortly after the year 2000, the industry began to recognize the importance of the relationships of applications and databases running independently and speak about what is today called *SOA (Service Oriented Architecture)*. This is delineated from N-tier client-server by the absence of a common DBMS; SOA services each have the own DBMS whereas a client-server system shares a common DBMS, even if the common DBMS is distributed across processes and/or processors.

Foundations

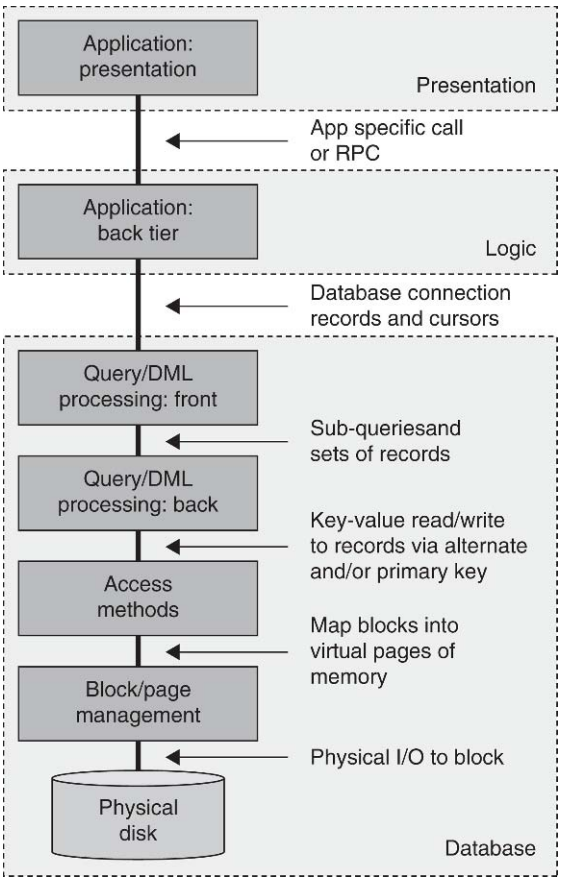
To understand the process and processor architecture of DBMS and applications, the reader first looks at a sketch of their high-level architecture independent of the processes and processors implementing the components of the architecture, followed by some common patterns for breaking this work up.

Layers of the DBMS and Application

Figure 2, depicts a breakdown of both the 3-tier application and the underlying DBMS. The dotted lines show the classic 3-tier architecture. The *presentation tier* is connected to the *logic tier* with an application specific call or RPC. The *logic tier* is connected with the *database tier* using a database connection.

Inside the database, more layers of abstraction are utilized to examine different process and processor architectures. Closest to the application is the front-end to the *Query and DML Processor*: it accepts a database connection from the application and is responsible for the processing DML and queries and returning the results across the connection. The front-end of this function will interact with a back-end which is intimate with the various access methods which hold the database records and/or alternate keys. The back-end of the Query and DML Processor will interact with the *Access Methods* using keys and records. The Access Methods, in turn, use blocks (which are mapped into in-memory pages) by the *Block and Page Manager*, which issues physical I/Os to the disks themselves.

In the simplest DBMS architectures, the *Shared-Everything* design, all DBMS components are resident on the same computer system and they interact through memory. In the *Shared Disk* architecture, the interactions between the *Block and Page Management*



Process Structure of a DBMS. Figure 2. The architecture of a 3-Tier application and its supporting database management system. The notations beside the arrows denote the formats of the requests and data flowing between the layers.

and the *Physical Disk* are spread apart. Finally, the *Shared Nothing* architecture separates the front and back ends of the Query and DML Processor.

Client-Server Computing

When client-server computing first arrived on the scene, its hallmark was the separation of the application from the DBMS itself leaving the database on its own system. Initially, this was done with two tiers, the client and DBMS-server. The client interacted with the server using a database connection such as ODBC (See Fig. 3).

The transition between two-tier client-server systems and three-tier (or even N-tier) client-server systems lies in the architecture of the application itself. It is simply the splitting up of the application tiers that differentiate these (See Fig. 4).

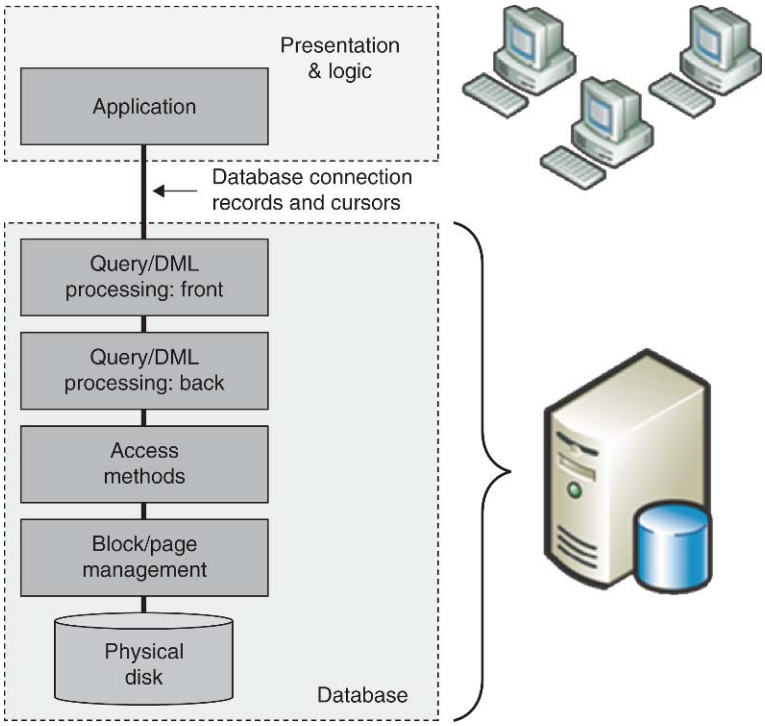
As soon as the 3-tier architecture was introduced, new challenges arose in the management of transactions as they propagated through the system. In 2-tier client-server architectures, the transactional scope was bounded by a time interval on the database connection. Now, in a 3-tier client-server scheme, the work initiated by the Presentation layer may need to be atomic as it propagates through different servers implementing the Logic layer. For the moment, consider this architecture with a single process DBMS at the back-end (See Fig. 5).

New notions in the management of transaction identifiers needed to be created for this to work. The identity of the atomic transaction needed to be propagated with the RPC or other app-specific call from the client to the middle-tier server. Also, the same transaction-id could now arrive at the database from *different* database connections. A window of time using the database-connection could no longer be a surrogate for the transaction. These challenges came as the application that related to the database underwent changes in its process architecture.

Multi-Database Computing

As distributed systems progressed, there were occasions in which a 2-tier client wished to do transactional work across multiple database servers. This was only possible with the arrival of BOTH distributed transactions (implemented with two-phase commit) AND the ability for the client application to create a transaction independent of the database connection and manage the association of the transaction to the connections to the separate databases. This involved both extensions to the client libraries and the creation of DBMS server to DBMS server distributed transaction management.

Most systems did not implement transactional support for multiple databases in 2-tier applications until after this was accomplished for 3-tier applications. The need to have a client call different middle-tier application servers with different databases was the pressure that led to the implementation of two phase commit for distributed transactions across the multiple databases. Only later were the facilities for sharing transactions across database connections directly connected to a single client implemented. See Fig. 6 for a depiction of a 3-tier client-server application where some of the middle-tier servers use different databases all associated with a single common transactional scope.



Process Structure of a DBMS. Figure 3. Two-tier client-server architecture. The application is separated out from the database. The interaction is maintained with a database connection.

Service-Oriented Architectures (SOA)

A recently popular application architecture is the service-oriented architecture. It is distinguished from the client-server application architecture in that there are no transactions shared across the service boundaries and, indeed, no direct visibility to the partner's database. All access to the database is indirect and mitigated by the application. Because there are no shared transactions across the service boundaries and there are no database connections or semantics across these boundaries, the database management system does not see the SOA architecture.

SOA implementations often use an internal client-server architecture to build out a single service. Still, from the standpoint of the process architecture of a DBMS, the use of an application combined with its database in a larger SOA system is not of any impact on the DBMS process structure.

Shared-Memory (Single Database) DBMS Architectures

In a shared-memory (or sometimes called shared-everything) system, the DBMS runs in a single process or, at least, in a fashion where the processes are able to share their memory. Figure 1 is an example of a shared-memory system, one where multiple processes have a special mechanism to access special (protected) memory. The architecture depicted in Fig. 1 is not a client-server system.

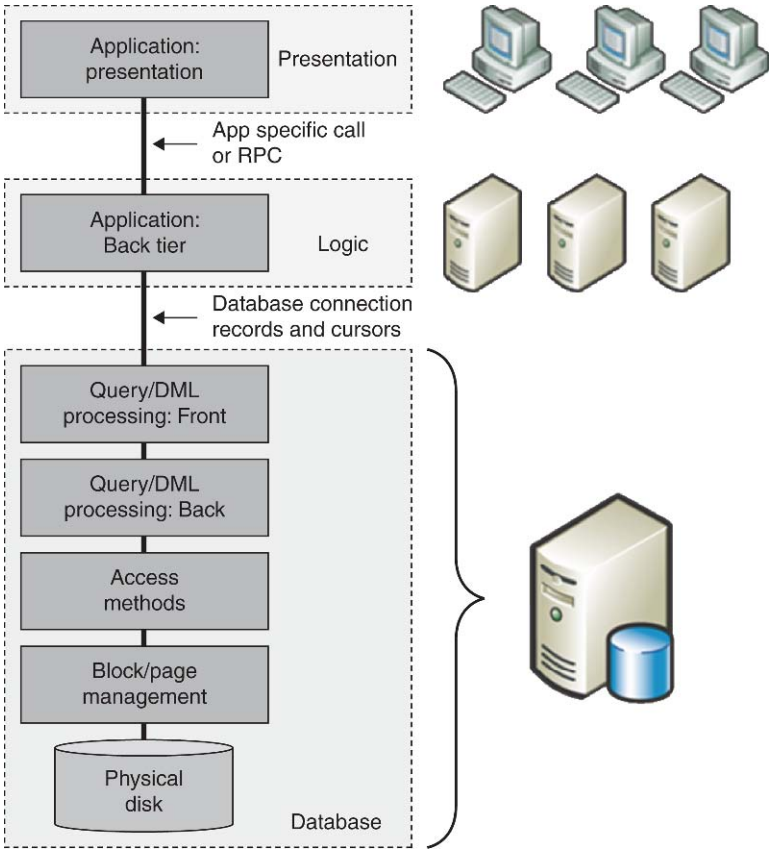
Most shared-nothing systems are a little less exotic and look like Figs. 3–5. There is a single database and it runs on a large process (perhaps with multiple threads within a shared-memory multiprocessor).

Figure 6 is considered to be a shared-everything system *precisely because each database is a separate database*, and there cannot be queries issued across them. There is no way to view the data from these multiple databases except through the assistance of application logic. There cannot exist a single database connection to that pool of databases – they are separate databases.

The presence of distributed transactions spanning multiple databases does not make them a single database.

Shared-Disk (Single Database) DBMS Architectures

Shared-Disk systems comprise a cluster in which multiple independent machines have access to a pool of disks. In addition to some mechanism for sending messages across the machines, each of the computers



Process Structure of a DBMS. Figure 4. 3-tier client-server architecture. The presentation, logic, and database are distributed across different systems.

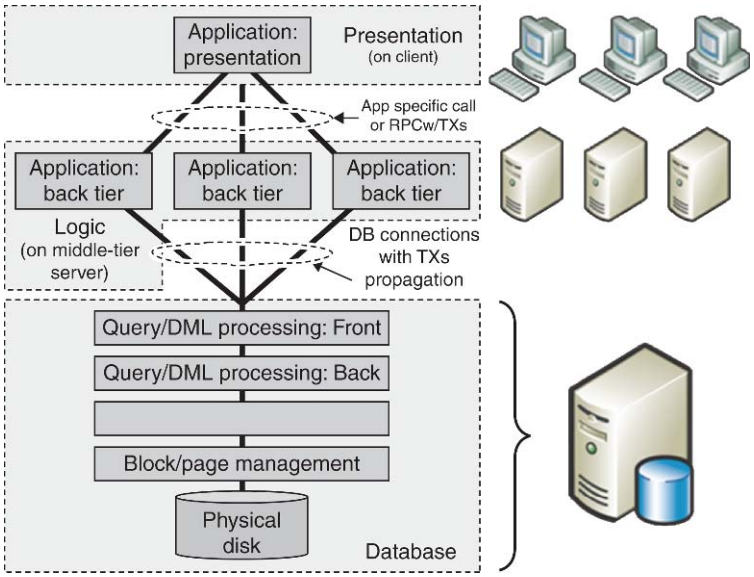
in the cluster can read and write pages of disk across a collection of many disk spindles. Typically, this is implemented with some form of disk controller managing access by the computers to the disks. Figure 7 shows a sketch of the hardware architecture of a shared-disk cluster.

The distinction between the database process architecture versus the application process architecture is sometimes confusing. The first shared-disk systems used block mode terminals and there was no notion of a smart client at that time. As an example, consider Fig. 8 which is a slight modification of Fig. 7. Figure 8 depicts a system in which the business logic (the middle tier of a three tier system) runs on the same processors (or potentially the same processes in the same spirit as shown in Fig. 1). In this example, the business logic of the application as well as the entire processing of the DBMS can run in the same process (or at least the same processor) while still scaling across a multiprocessor cluster with tremendous efficiency if the load characteristics are appropriate for the architecture.

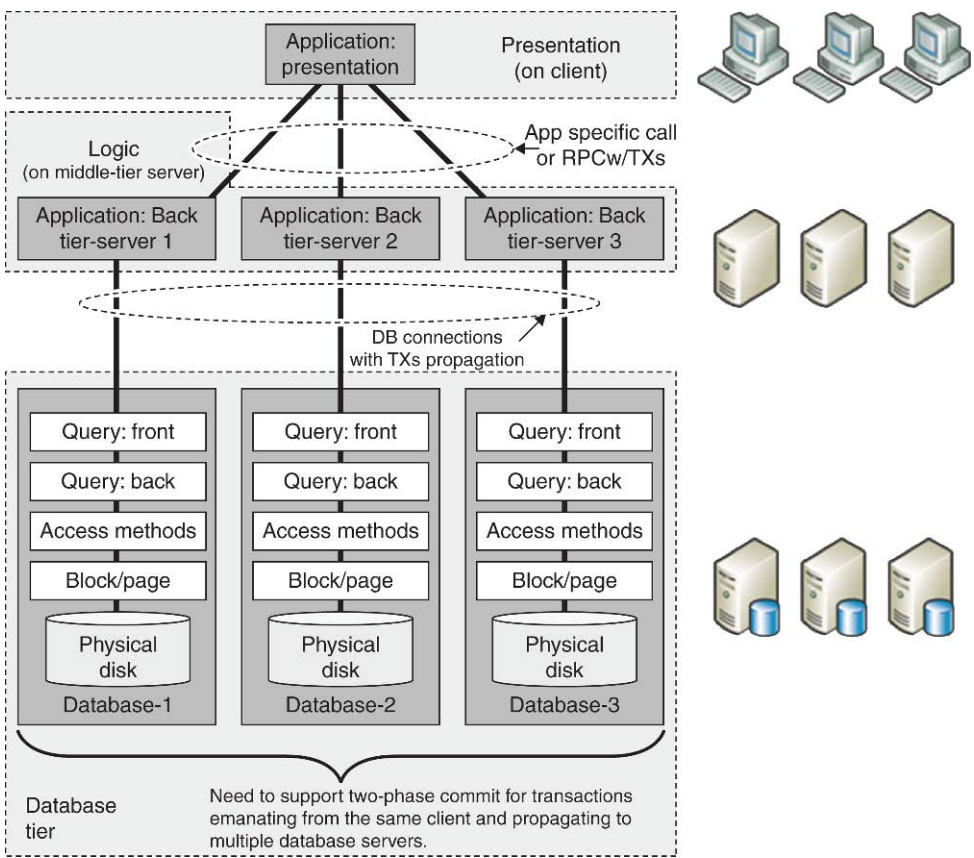
Shared-Disk DBMS systems have the advantage that, once all of the blocks are brought into memory, the entire query can be processed within a single process. This style of distributed implementation delivers high performance when the workload splits into easily partitioned sets of blocks but works less efficiently when the workload has conflicts over the blocks needed by the different processes.

Shared-Nothing (Single Database) DBMS Architectures

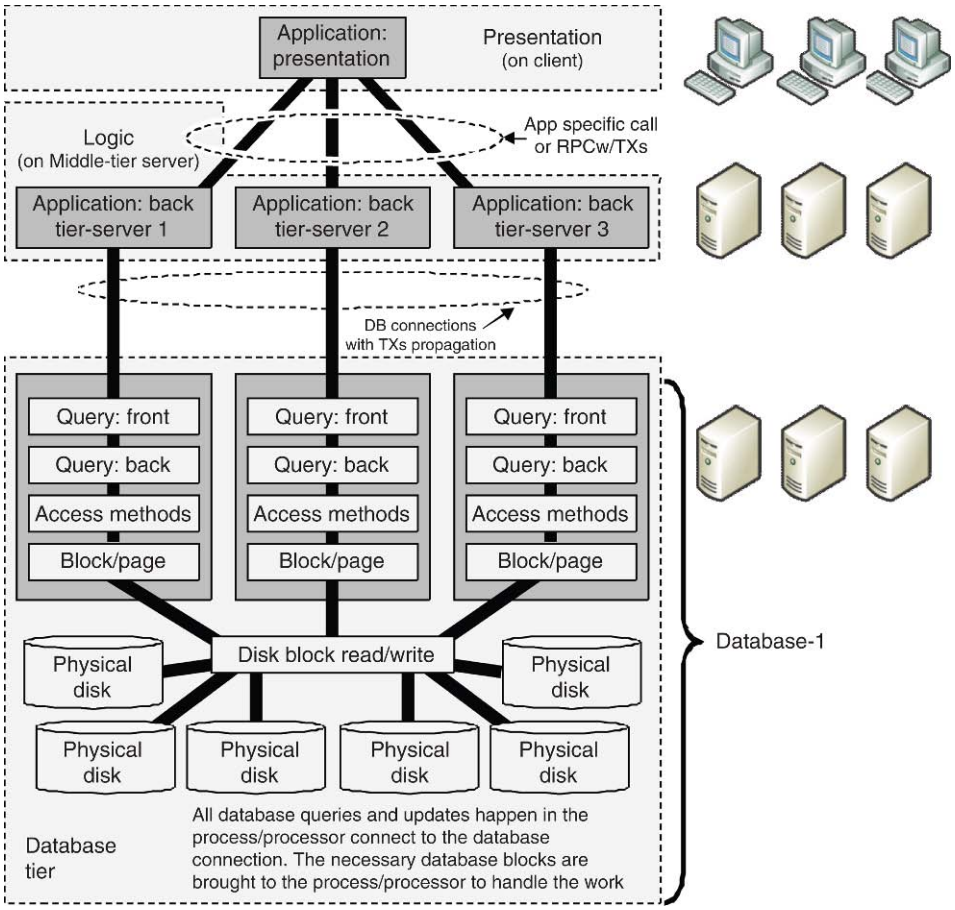
Share-Nothing DBMS architectures are implemented by splitting the query processing engine into a front-end and a back-end. Messaging is used to pass portions of the query or update from the front-end to the back-end. Resulting sets of records are passed from the back-end to the front-end where the completion of the query is performed. The optimizations used in these architectures have been the source of significant research and engineering and can result in fascinating performance gains. Figure 9 shows a Shared-Nothing DBMS in which the application's Business Logic is



Process Structure of a DBMS. Figure 5. Building a 3-tier client-server application necessitates both the management of the same transaction coming into the DBMS on different database connections AND the propagation of transactions from the client to the middle-tier server.



Process Structure of a DBMS. Figure 6. A 3-tier client-server application with DIFFERENT databases not only has to manage the propagation of transactions from client to app-server AND app-server to DBMS, it must in addition manage the atomic two-phase commit of the transactions across database servers.

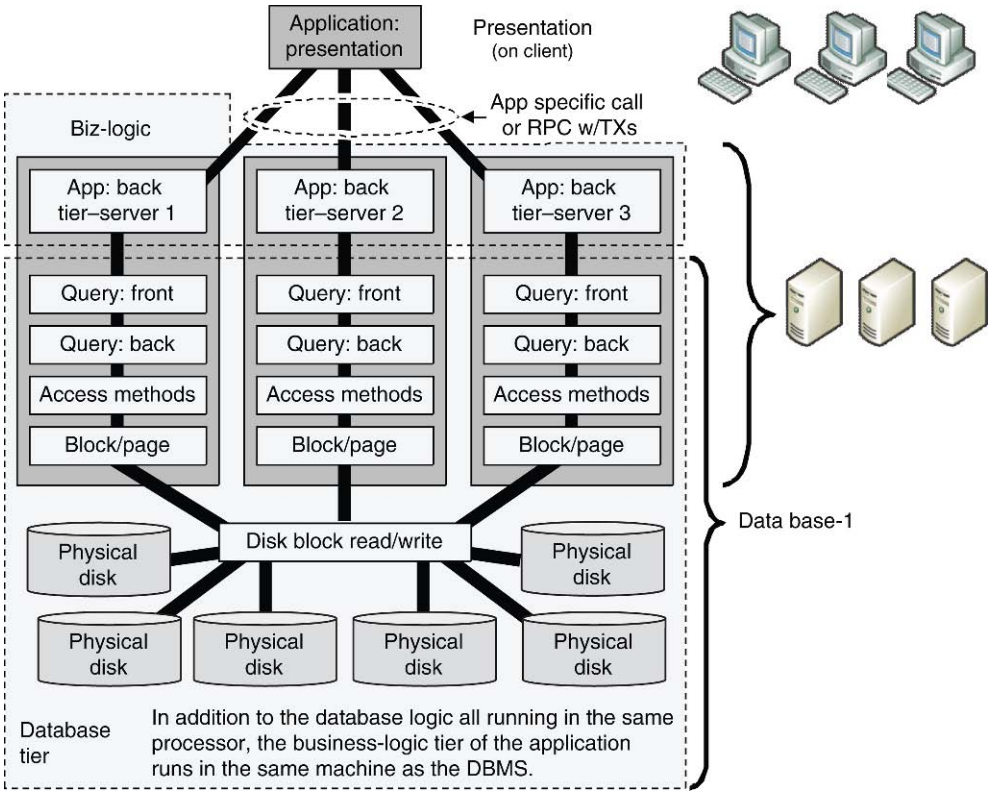


Process Structure of a DBMS. Figure 7. A Shared-Disk Database Management System. For the first time in our figures, the single database SPANS multiple processes and, indeed, multiple processors. The single database (with single database semantics presented to the application) runs on many different computers but sharing the access to the physical disks. Special locking infrastructure on the contents of the blocks of disk must be maintained.

running on the same scale out cluster as the DBMS. Just like the Shared-Disk DBMS, it is important to realize that the application's process/processor architecture may take different forms. Running the Business Logic close to the Front-End of the DBMS is one configuration of Shared-Nothing.

Implications of Shared-Nothing, Shared-Disk, and Shared-Everything Architectures

For a number of years, there have been debates in the industry about the strengths and weaknesses of different process architectures for a DBMS system. Before even engaging in these, it is important to remember the delineation of the DBMS architecture from that of a Service Oriented Architecture and, also, from the application architecture of an N-tier system. The term DBMS is used to refer to a single collection of records across which relational operations may occur. Service Oriented Architectures (SOAs) offer an aggregation of computation connected by business logic without the presence of spanning relational operations or transactions. N-tier application environments frequently offer atomic transactions across different databases but do not offer relational operations across the contents of the databases. So, the taxonomy of DBMS process structures refers only to the portion of the system providing a single relational database semantic.

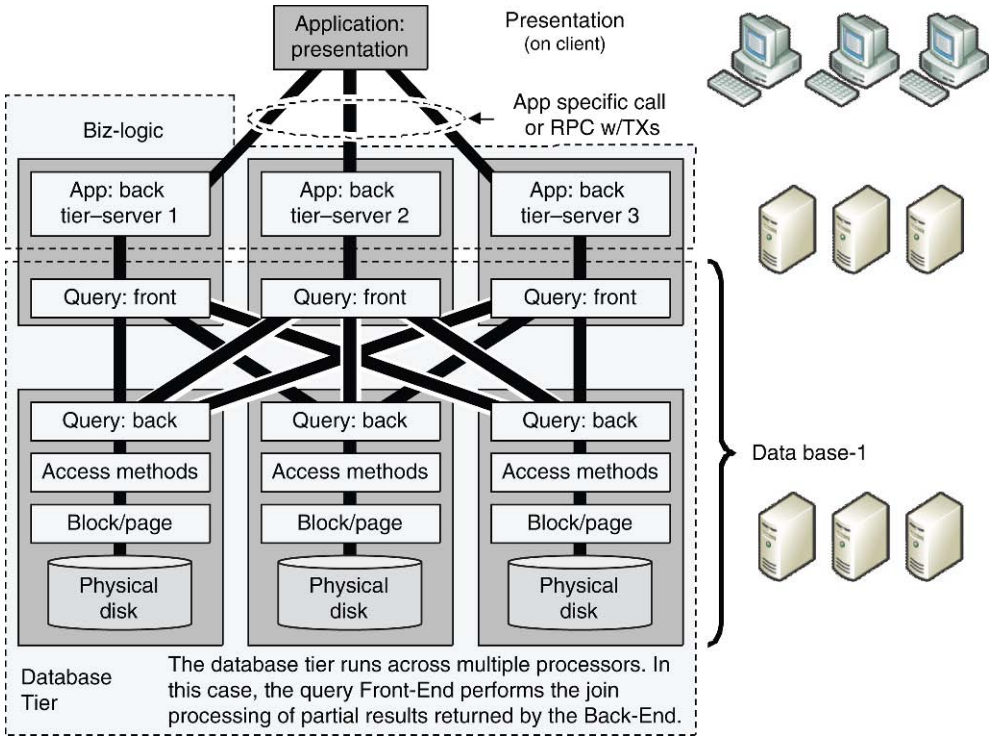


Process Structure of a DBMS. Figure 8. Shared-Disk DBMS within a Two-Tier application architecture.

The *Shared-Everything* DBMS architecture is by far the highest performing architecture in that it offers the most throughput for a single computer (it is not necessarily the most scalable one; scalability is discussed below). Shared-Everything DBMSs do not need to perform any messaging or other cross process communication because they don't cross processes in their implementation of the DBMS. All of the work is done within a single machine. Included in this architecture are various single memory multi-processor implementations. There are few concerns in a Shared-Everything DBMS about the types of queries and usage patterns because everything coexists in a single shared process. Again, Shared-Everything works wonderfully until the DBMS grows too large to fit into one system.

A *Shared-Disk* DBMS architecture allows for additional sharing by letting multiple DBMS processes and processors to access the same physical disks. When the usage pattern of the application has a low probability of conflict in its updates of a shared page, Shared-Disk DBMS systems are very efficient. The pages needed for a query are brought into the processor requesting them and the work of the application's transaction is handled inside one processor of the cluster in what is (hopefully) a very efficient fashion. When the data in question has low update rates, this typically works well. When a single data item is rapidly updated (called a "hot-spot"), this can lead to performance conflicts which cause the block of the database to be pulled back-and-forth across processors. Another challenge occasionally presented by Shared-Disk DBMS systems lies in block (or page) mode locking. Distinct records in the same block may observe lock conflicts causing performance challenges that would not be present in other architectures. Still, Shared-Disk DBMS systems have been wildly successful for many applications and allow many databases to scale beyond what a single shared-memory system could offer.

Shared-Nothing DBMS architectures typically carry a heavier cost to set up a complex transaction but can, in some cases, offer greater throughput over a particular amount of data. It is typical for a Shared-Nothing system to offer record locking. For systems with very high throughput "hot-spots," the data for the



Process Structure of a DBMS. Figure 9. A Shared-Nothing DBMS running with an application configuration which places the business logic tier on the same cluster as the Shared-Nothing DBMS. Also, in this configuration, the Query Front-End of the DBMS runs on the same processors as the Business-Logic.

"hot-spot" does not move around the cluster and it is possible for the system to get more transactions over the same piece of data in a fixed period of time. This is sometimes referred to as "moving the operation to the data" (Shared-Nothing) rather than "moving the data to the operation" (Shared-Disk).

When a database can fit into a single shared-memory multiprocessor, Shared-Everything offers distinct advantages (at least until the application's demands exceed the single machine and you have a problem). For databases that exceed this size and yet want full database semantics, there is a lively debate within the community about which architecture is better and different applications offer different performance characteristics.

Key Applications

Database process structures are an essential part of scaling past a single system. As discussed, the distinction between an application multi-processor architecture and a DBMS multi-processor architecture has many nuances.

Cross-references

- ▶ Application Server
- ▶ Client-Server Architecture
- ▶ Clustering
- ▶ Database Client
- ▶ Distributed Concurrency Control
- ▶ Distributed Database Systems
- ▶ Distributed DBMS
- ▶ Distributed Query Processing
- ▶ Distributed Transaction Management
- ▶ Multi-Tier Architecture
- ▶ ODBC
- ▶ Service Oriented Architecture
- ▶ Shared-Disk Architecture
- ▶ Shared-Memory Architecture
- ▶ Shared-Nothing Architecture
- ▶ Two-Phase Commit
- ▶ Multi-Tier Architecture

Recommended Reading

1. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, San Mateo, CA, 1992.

2. Michael S. (UC Berkeley). The case for shared nothing architecture. Database Eng., 9(1):4–9, 1986.
3. Oracle RAC (Real Application Clusters). http://www.oracle.com/database/rac_home.html
4. Susanne E., Jim G., Terrye K., and Praful S. A Benchmark of NonStop SQL Release 2 Demonstrating Near Linear Speedup and Scaleup on Large Databases. In Proc. 2000 ACM SIGMETRICS Int. Conf. on Measurement and Modeling of Comp. Syst., 1990, pp. 24–35.
5. The Tandem Database Group. NonStop SQL: a distributed high performance, high availability implementation of SQL. In Proc. of 2n High Performance Transaction Processing Workshop, 1989.

Processing Overlaps

Georgina Ramírez
Yahoo! Research Barcelona, Barcelona, Spain

Synonyms

Removing overlap; Controlling overlap

Definition

In semi-structured text retrieval, processing overlap techniques are used to reduce the amount of overlapping (thus redundant) information returned to the user. The existence of redundant information in result lists is caused by the nested structure of semi-structured documents, where the same text fragment may appear in several of the marked up elements (see Fig. 1). In consequence, when retrieval systems perform a focused search on this type of document and use the marked up elements as retrieval objects, very often result lists contain overlapping elements. In retrieval applications where it is assumed that the user does not want to see the same information twice, it may be necessary to reduce or completely remove this overlap and return a ranked list of no overlapping elements. Thus, depending on the underlying user model and retrieval application, different processing overlap techniques are used in order to decide, given a set of relevant but overlapping elements, what are the most appropriate elements to return to the user.

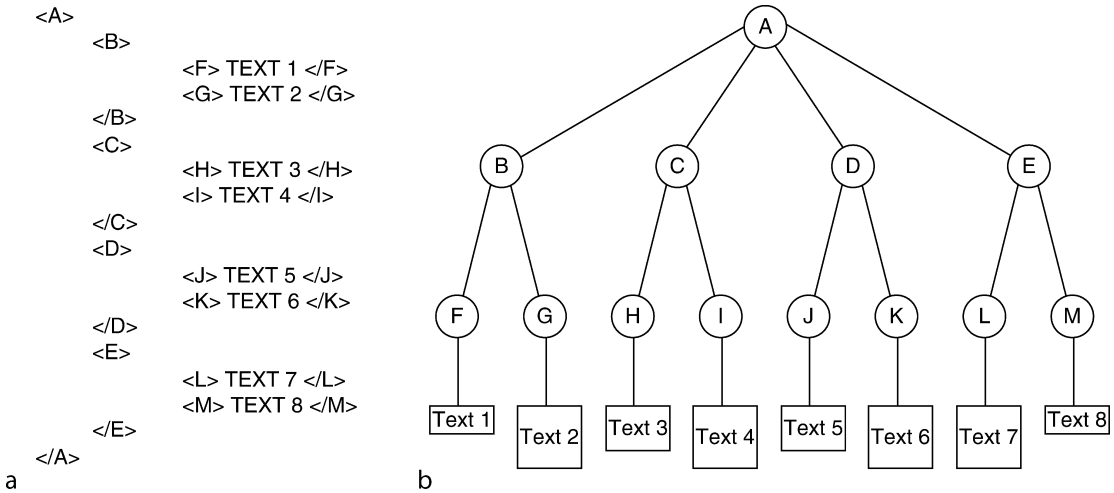
Historical Background

Although the problem of overlap in semi-structured text retrieval is as old as the semi-structured documents themselves, not much work has been published on processing techniques for reducing or removing overlap. Some related work can be found in the area of passage retrieval, where approaches that use a varying window size for passage selection might produce result lists with overlapping passages. However, most of this work is performed on unstructured documents and the approaches taken for processing overlap tend to be simpler.

In the domain of semi-structured documents, there are several areas where different overlap issues are studied. For example, there is quite some work in the area of evaluation of XML systems that addresses the so called overlap problem (e.g., [3]). A different overlap problem is created by the possibility that standards like SGML provide of having multiple annotations (markups) on the same document (a.k.a. multiple hierarchies). In this case the overlap is produced by the structure of the different annotations. Since the multiple hierarchies complicate the use of standard retrieval techniques on this type of documents, work on this area is still focusing on addressing other indexing and retrieval issues. It is only recently that in the domain of XML documents, several approaches have been presented that address the problem of processing overlap from result lists containing overlapping elements. The next section summarizes some of them.

Foundations

One of the advantages of semi-structured documents is that retrieval systems can perform focused search by simply using the marked up divisions of the documents (elements) and retrieving those instead of the whole documents. However, since elements overlap with each other (see Fig. 1), when using traditional ranking techniques to independently rank these elements, result lists often contain many overlapping elements. This is due to the nested structure of semi-structured documents, where the same text fragment is usually contained in several of the marked up elements. Thus, when a specific element is estimated relevant to the query, all the elements containing this element (a.k.a. ancestors) will also be estimated to some degree relevant to the query. Furthermore, this element is probably estimated relevant because it contains several relevant elements (a.k.a. descendants). For example, if a section of a document is estimated highly relevant, it probably contains several highly relevant paragraphs and it is contained in a relevant article. If all of these elements are returned to the user, the amount of redundant information contained in the result list will be considerable. In retrieval scenarios where users do not like to see the same information twice, retrieval systems



Processing Overlaps. Figure 1. Example of a semi-structured document and its tree structure representation. Note that each fragment of the document is contained in three different elements (nodes in the tree).

need to decide which of these relevant but overlapping elements is the most appropriate piece of information to return to the user. The final decision on which elements the system should return depends on the search application and the underlying user model but a common goal is to reduce redundancy in the result lists. Although this can be done at indexing time (e.g., by selecting a subset of non-overlapping elements as potential retrievable objects), commonly this is done by removing overlapping elements from the result set, after retrieval systems have produced an initial ranking of all elements. Processing overlap techniques have recently been widely discussed in the domain of XML retrieval, where different approaches have been presented. The rest of this section presents and discusses some of them.

Using Element Types

A simple way to reduce overlap is to select a subset of element types and consider only these for retrieval. For example, if sections of documents are considered to be the most appropriate pieces of information, retrieval systems might want to use only these as retrievable objects and ignore all the other element types. This can be done at indexing time or by post-filtering the result lists. Depending on the number and element types selected, overlap is reduced at different degrees. Note that it might not always be possible to completely remove overlap; even if a single element type is selected as a unique retrievable object, it is still possible that elements of the same type overlap each other. The main drawback of this approach is that, since it is desirable to select element types that are likely to be relevant and useful to the user, it requires knowledge of the structure of the documents and its common usage.

Using Paths

A common approach for processing overlaps keeps the highest ranked element on each path and removes its ancestors and descendants from the result list, i.e., all the elements in the result list that contain or are contained within it (e.g., [6], [2]). It is important to notice that depending on the order in which the different paths are processed different outputs might be produced.

In [6] the authors present a two steps algorithm to remove the overlap. The first step is used to select the highest scored element from each relevant path. Since their algorithm selects the elements from the different paths simultaneously, the output may still contain overlapping elements. That is why a second step is needed, to completely remove overlap. This is done by selecting again (this time from the output of the first step) the highest scored element from each path:

Algorithm 1

1. Select highest scored element from each relevant path.
2. Select highest scored element from each relevant path in output of step 1.

Another common way to remove overlap using paths [5] is to recursively process the result list by selecting the highest ranked element and removing any element from

lower ranks that belongs to the same path (it contains the selected element or it is contained within it):

Algorithm 2

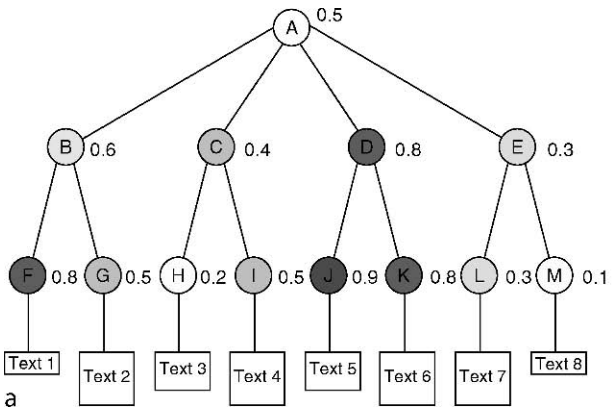
1. Return highest ranked element from result list.
2. Remove from result list all the elements belonging to the same path.
3. Repeat step 1 and 2 until result list is empty.

The underlying assumption of this type of approaches is that the most appropriate piece of information in each path has been assigned a higher score than the rest and therefore, removing overlap is simply a presentation issue. These approaches rely completely on the underlying retrieval models to produce the best ranking. This could indeed be the case if the retrieval model would consider, when ranking, not only the estimated relevance of the element itself but also its *appropriateness* compared to other elements in the same path. However, since many retrieval models rank elements independently, the highest scored element may not be the most appropriate one, i.e., the one the user prefers to see.

To illustrate the different outputs of the previous algorithms, have another look at the example document from Fig. 1. Imagine now that, given a query, the retrieval model estimates the relevance of each element in the document. Figure 2 shows the retrieval scores obtained by each of the elements and the outputs produced when removing overlap with the algorithms described above. Both algorithms produce a result list of non-overlapping elements. However, there are substantial differences. The main drawback of the first algorithm is that it might miss some relevant information. For example, one could argue that element K should also be contained in the output list. To be able to do that, the algorithm should consider structural relationships between elements and re-add element K to the result list when it decides to remove element D in the second step.

Although the second algorithm produces a more complete list, someone could argue that the output produced is not the most desirable. For example, imagine that elements F and G are, respectively, the title and the abstract of the document. In this case, even if the title has been ranked high (it may contain most or all of the query terms), users might prefer to see a result item containing both, title and abstract (i.e., element B) instead of seeing both elements independently. In general, it can be argued that retrieval systems should only return those elements that have enough content information to be useful and can stand alone as independent objects. A similar example can be seen for elements D, J, and K. Even if J is ranked higher, element D contains mostly relevant information and therefore, it might be more desirable (from a user perspective) to read element D, than J, and K independently. Furthermore, for elements E, L, and M it could be argued that it is better to return L than E because the relevance estimated for element E is due to the content of L and no extra benefit is obtained when returning E.

In these, cases a better output might be produced if the algorithms would consider structural relationships between elements when deciding which elements to return to the user.



b

Original output: <J, D, F, K, B, A, G, I, C, E, L, H, M>

Algorithm 1:
Output step 1: <J, F, D, B, A>
Final output: <J, F>

Algorithm 2:
Final output: <J, F, K, G, I, E, H>

Processing Overlaps. Figure 2. Example of a retrieval run and the resulting outputs when removing overlapping elements with algorithm 1 and 2. The numbers on the tree indicate the initial retrieval scores obtained by each of the elements.

Using Structural Relationships for Re-Ranking

The following approaches present more advanced techniques that exploit the structural relationships within a document to decide which elements should be removed or pushed down the ranked list and which ones should be returned to the user (e.g., [5], [1], [4]). These techniques often modify the initial ranking predicted by the retrieval model.

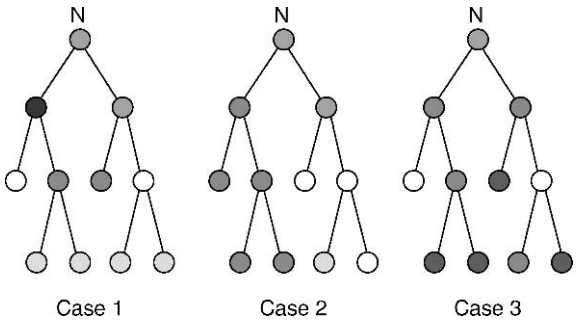
In [1] elements are re-ranked by adjusting the element scores of the lower ranked elements according to their containment relationship with other higher ranked elements. The assumption underlying this approach is that the score of the elements that are contained or contain other elements that have been already shown to the user (i.e., are ranked higher) should be adjusted in order to reflect that the information they contain might be redundant. They do that by reducing the importance of terms occurring in already reported elements. The basic algorithm is similar to algorithm 2 but instead of removing the ancestors and descendants of the reported elements, their scores get adjusted:

Algorithm 3

1. Report the highest ranked element.
2. Adjust the scores of the unreported elements.
3. Repeat steps 1 and 2 until m elements are reported.

The author also presents an extended version of the algorithm where different weighting values are used for ancestors and descendants and where the number of times an element is contained within others is considered. For example, a paragraph contained in an already seen section and in an already seen article is further punished because the user has already seen this information twice. Note that this algorithm is not designed to remove the overlap but to push down the result list those elements that contain redundant information.

In [4] a completely different approach is taken. The authors present a two step re-ranking algorithm for removing overlap. The first step identifies clusters of highly ranked results and picks the most relevant element from each cluster. The second round is used to remove any remaining overlap between the selected elements. The selection criteria for the first step is based on three different cases (illustrated in Fig. 3): (i) if an element N has a descendant that is substantially more relevant, the element N is removed from the result list, (ii) if case 1 does not hold and the element N has a child that contains most of the relevant information (the relevant elements are concentrated under this child), the element N is also removed from the result list, and (iii) if none of the previous cases hold and the results are evenly distributed under the element N, then the element N is kept and all its descendants removed from the result list. In the rest of the cases they do not do anything and leave the final overlap removal for the second phase. In the second step, they remove overlap by comparing the score of each element with the ones of its descendants. If the score of the element is bigger, all the descendants are removed and the element is kept. Otherwise, the element is removed.



Processing Overlaps. Figure 3. Illustration of the three cases considered in the approach presented in [4]. The grayer the node, the higher the relevancy estimated for that node.

In [5] the authors present an approach that makes use of an *utility* function that captures the amount of *useful* information contained in each element. They argue that to model the *usefulness* of a node three important aspects need to be considered: (i) the relevance score estimated by the retrieval model, (ii) the size of the element, and (iii) the amount of irrelevant information the element contains. They present an algorithm that selects elements according to the estimated *usefulness* of each element. If an element has an estimated *usefulness* value higher than the sum of the *usefulness* values of its children, then the element is selected and the children are removed. Otherwise, the children elements whose *usefulness* value exceeds some threshold are selected and the element is removed.

Key Applications

Processing overlap techniques are needed in any retrieval application where users need to be directed to specific parts of documents and there are not predefined retrieval units.

Experimental Results

For every presented approach, there is an accompanying experimental evaluation in the corresponding reference. Commonly, the referred article provides an overview of the performance of the approach and of its variations. However, it is difficult to use the reported evaluations to compare approaches between articles. The main reason is that all approaches make use of different retrieval models for their initial runs, thus it is not clear whether the performance obtained after removing overlap is due to the underlying retrieval model or to the approach used to remove the overlap. Besides, not all the presented approaches experiment on the same dataset or use the same evaluation measures to report their numbers.

In [5] the authors present an experimental comparison between several of the approaches described above. They show that the best performing approach is the one that returns only paragraphs. As a general trend for the first type of approach (the ones that select a specific element type), the longer the element type selected, the worse the performance. The authors also show that their approach of estimating the *usefulness* of an element can help to improve retrieval performance (in terms of precision at low recall levels) when compared to the approach of using paths (algorithm 2).

Data Sets

Since 2005 the *INitiative for the Evaluation of XML Retrieval* (INEX) provides a data-set that can be used to test processing overlap strategies (see http://inex.is.informatik.uni-duisburg.de/).

Cross-references

- ▶ INEX
- ▶ Structured Document Retrieval
- ▶ XML Retrieval

Recommended Reading

1. Clarke C.L.A. Controlling overlap in content-oriented XML retrieval. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 314–321.
2. Geva S. GPX – gardens point XML IR at INEX 2005. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrievals, 2006, pp. 240–253.
3. Kazai G., Lalmas M., and de Vries A.P. The overlap problem in content-oriented XML retrieval evaluation. In Proc. 30th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2004, pp. 72–79.
4. Mass Y. and Mandelbrod M. Using the INEX environment as a test bed for various user models for XML retrieval. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrievals, 2006, pp. 187–195.
5. Mihajlovi V., Ramírez G., Westerveld T., Hiemstra D., Blok H.E., and de Vries A.P. TIJAH scratches INEX 2005: vague element selection, image search, overlap and relevance feedback. 2006, pp. 72–87.
6. Sauvagnat K., Hlaoua L., and Boughanem M. XFIRM at INEX 2005: ad-hoc and relevance feedback tracks. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrievals, 2006, pp. 88–103.

Processing Structural Constraints

Andrew Trotman
University of Otago, Dunedin, New Zealand

Definition

When searching unstructured plain-text the user is limited in the expressive power of their query – they can only ask for documents that are about something. When structure is present in the document, and with a query language that supports its use, the user is able to write far more precise queries. For example, searching for "smith" in a document is not necessarily equivalent to searching for "smith" as an author of a document. This increase in expressive power should lead to an increase in precision with no loss in recall. By specifying that "smith" should be the author, all those instances where "smith" was the profession will be dropped (increasing precision), while all those in which "smith" is the author will still be found (maintaining recall).

Historical Background

With the proliferation of structured and semi-structured markup languages such as SGML and XML came the possibility of unifying database and information retrieval technologies. The Evaluation of XML Retrieval (INEX) was founded in 2002 to examine the use of semi-structured data for both technologies. It was expected that the use of structure would not only unify the two technologies but would also improve the performance of both.

Foundations

User Querying Behavior

When using an information retrieval search engine the user typically has some information need. This information need is expressed by the user as a keyword query. There are many different queries that could be drawn from the same information need. Some might contain only keywords, others phrases, and others a combination of the two. It is the task of the search engine to satisfy the information need given the query. Because the task is not to satisfy the query, the terms in the query can be considered nothing more than hints by the user on how to identify relevant documents. It is likely that some relevant documents will not contain the user's keywords, while others that do might not be relevant.

If the user does not immediately find an answer they will often change their query, perhaps by adding different keywords, or by removing keywords. If the query syntax permits they might add emphasis markers (plus and minus) to some terms.

With a few exceptions semi-structured search engines remain experimental, so user behavior cannot be studied in a natural environment. Instead the user behavior is expected to mirror that of other search engines, or search engines that include some structural restriction.

The model used at INEX is that a user will give a query containing only search terms, then, if they are dissatisfied with the results, they might add structural constraints to their query. The keyword searches are known as Content Only (CO) and when structure is added as Content Only + Structure (CO + S) or Content And Structure (CAS) queries. Just as the keywords are hints, so too are the structural constraints. For this reason they are commonly referred to as structural hints.

The addition of structure to an otherwise content only search leads to a direct comparison of the performance of a search engine before and after the structural constraint has been added. The two queries are instantiations of the same information need, so the same documents or document components are relevant to each query making a direct comparison meaningful.

The analysis of runs submitted to INEX 2005 (against the IEEE document collection) showed no statistical difference in performance between the top CO and top CO + S runs – having structural hints in the query did not improve performance [12]. Even at low levels of recall (1 and 10%) no significant improvement was seen. About half the systems showed a performance gain, the other half no gain.

There are several reasons why improvements are not seen: first it could be a consequence of the structure present in the IEEE collection; second (and more likely) it could be that users are not proficient at providing structural hints.

The result was backed up by a user study [13] in which users were presented with three ways of querying the document collection: keywords, natural language (including structure), and Bricks [15] (a graphical user interface). Sixteen users each performed six simulated work tasks, two with each interface. The same conclusion is drawn, that is no significant improvement was seen when structure was used in querying.

Structural Constraints

There are two reasons a user might add structural constraints to a query. The first is to constrain the size of the result. When searching a collection of textbooks it is, perhaps, of little practical use to identify a book that satisfies the user need. A better result might be a chapter from the book, or a section from the chapter, or even a single paragraph. One way to identify the best granularity of result is to allow the user to specify this as part of the query. These elements are known as target elements.

The user may also wish to narrow the search to just those parts of a document he or she knows to be appropriate. In this case the user might search for "smith" as an author in order to disambiguate the use from that as any of: an author, a profession, a street, or a food manufacturer. Restricting a query to a given element does not affect the granularity of the result; instead it lends support on where to look so such elements are known as support elements. Both target elements and support elements can appear in the same query.

It is not at all obvious from a query whether or not the user expects the constraint to be interpreted precisely (strictly) or imprecisely (vaguely). In the case of "smith" as an author, it is likely that "smith" as a profession is inappropriate, but "smith" as an editor might be appropriate. If the target element is a paragraph, then a document abstract (about the size of a paragraph) is likely to be appropriate, but a book not so.

The four possible interpretations of a query were examined at INEX 2005 [11]. Runs that perform well with one interpretation of the target elements do so regardless of the interpretation of the support elements. The interpretation of the target element does, however, matter. The consequence is that the search engine needs to know, as part of the query, whether a strict or vague interpretation of the target element is expected by the user.

Processing Structural Constraints

Given a search engine, the strict interpretation of target elements can be satisfied by a simple post-process eliminating all results that do not match. As just discussed above, strictly processing support elements has been shown to be unnecessary.

Several techniques for vaguely satisfying target element constraints have been examined including ignoring them, pre-generating a set of tag-equivalences, boosting the score of elements that match the target element, and propagating scores up the document tree.

Ignoring Structural Constraints

Structural constraints might be removed from the query altogether and a Content Only search engine used to identify the correct granularity of result.

Tag Equivalence

A straightforward method for vaguely processing structural constraints is tag equivalence. A set of informational groups are chosen *a priori* and all tags in the DTD (DTD is the Document Type Definition, specifying the format of the XML documents forming the collection.) are mapped to these groups. If, for example, <p> is used for paragraphs and <ip> is used for initial paragraphs, these would be grouped into a single paragraph group.

Mass and Mandelbrod [5] *a priori* choose appropriate retrieval units (target elements) for the document collection and build a separate index for each. The decision about which units these are is made by a human before indexing. A separate index is built for each unit and the search is run in parallel on each index. Within each index the traditional vector space model in used for ranking. The lexicon of their inverted index contains term and context (path) information making strict evaluation possible. Vague evaluation of paths is done by matching lexicon term contexts against a tag-equivalence list.

Mihajlović et al. [6] build their tag equivalence lists using two methods, both based on prior knowledge of relevance. For INEX 2005 they build the first list by taking the results from INEX 2004 and selecting the most frequent highly and fairly relevant elements and adding the most frequently seen elements from the queries. In the second method they take the relevant elements from previous queries targeting the same element and normalize a weight by the frequency of the element in the previous result set (the training data, in this case INEX 2004). Using this second method they automatically construct many different tag equivalence sets using the different levels of relevance seen in the training data.

In a heterogeneous environment in which many different tags from many different DTDs are semantically but not syntactically identical, techniques from research into schema-matching [1] might be used to automatically identify tag equivalence lists.

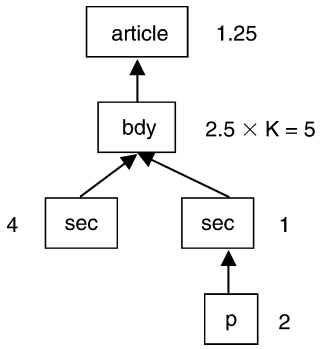
Structure Boosting

Van Zwol [14] generates a set of results ignoring structural constraints then boosts the score of those that do match the constraints by linearly scaling by some tag specific constant. The consequence is to boost the score of elements that match the structural constraints while not removing those that do not. A score penalty is also used for deep and frequent tags in the expectation of lowering the score of highly frequent (and short) tags. A similar technique is used by Theobald et al. [9] who use it with score propagation.

Score Propagation

Scores for elements at the leaves of the document tree (that is, the text) are computed from their content. Scores for nodes internal to the document tree are computed from the leaves by propagating scores up the tree until finally a score for the root is computed. Typically as the score propagates further up the tree, its contribution to the score of an ancestor node is reduced (see the entry on *Propagation* based structured text Retrieval for details).

Figure 1 illustrates score propagation. A search term is found to occur two times in the p element, and four times in the (left) sec element. With a decay factor of 0.5, the score of the bdy element is computed as 4 *0.5 + 2 * 0.5 * 0.5 = 2.5. The score for the article element is computed from that score likewise. If the target element is bdy and the score, for example, is boosted by $K = 5$, then the element with the highest



Processing Structural Constraints. Figure 1. Score Propagation, each time a score is propagated the score is weakened (in the example: halved), but at target nodes it is boosted (in the example: K = 2).

score is that element. The score for K and the propagation value are chosen here for illustrative purposes only and should be computed appropriately for a given document collection.

Hurbert [3] uses score propagation with structure reduction – if a node in the tree does not match a constraint in the query then the score there is reduced by some factor. In this way all nodes in the tree obtain scores but those matching the constraints are over-selected for. Sauvagnat et al. [8] use score propagation in a similar way but in combination with tag equivalence.

Key Applications

Retrieval of document components from structured document collections.

Future Directions

The best performing search engines that interpret structural constraints have not yet significantly outperformed those that ignore them. Several reasons have been forwarded.

There is evidence to suggest specifying a structural constraint is difficult for a user. Studies into the use of structure in INEX queries suggest that even expert users, when asked to give structured queries, give simple queries [12]. This is inline with studies that show virtually no use of advanced search facilities on the web.

Structure aware search engines are not as mature as web search engines and as yet the best way to use structural constraints (when present in a query) is unknown. The annual INEX workshop provides a forum for testing and presenting new methods.

Improvements were not seen when the IEEE document collection was used, but this result may not generalize to all collections. Alternative collections including newspapers, radio broadcast, and television, have been suggested [7,10]. In 2006, INEX switched to using the Wikipedia as the primary test collection for *ad hoc* retrieval, but a comparative study on that collection has not yet been conducted.

Relevance feedback including structural constraints has been examined. Users might provide feedback on both the desired content and the preferred target element, or just one of these. Evidence suggests that including structure in relevance feedback does improve precision.

Experimental Results

Evidence that use of structure increases precision is tentative. In the XML search engine of Kamps et al. [4], no significant difference is seen overall, however significant differences are seen at early recall points (the first few tens of documents). The search engine of Geva [2] recently performed better without structure than with.

At INEX 2005, a comparative analysis of performance with and without structural constraints on the same set of information needs was performed [1]. The best structure run was compared to the best non-structure run and no significant difference was found. Not even a significant difference at early recall points was found. On a system by system basis about half the search engines show a performance increase.

Cross-references

- ▶ Content-and-Structure Query
- ▶ Content-Only Query
- ▶ INitiative for the Evaluation of XML Retrieval
- ▶ Mean Average Precision
- ▶ Narrowed Extended XPath 1
- ▶ Propagation-based Structured Text Retrieval
- ▶ XML Retrieval

Recommended Reading

1. Doan A. and Halevy A.Y. Semantic integration research in the database community: a brief survey. AI Magazine, 26(1): 83–94, 2005.
2. Geva S. GPX – gardens point XML IR at INEX 2006. In Proc. 5th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2007, pp. 137–150.

3. Hubert G. XML retrieval based on direct contribution of query components. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 172–186.
4. Kamps J., Marx M., Rijke M.D., and Sigurbjörnsson B. Articulating information needs in XML query languages. Trans. Inf. Sys., 24(4):407–436, 2006.
5. Mass Y. and Mandelbrod M. Using the INEX environment as a test bed for various user models for XML retrieval. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 187–195.
6. Mihajlovic V., Ramírez G., Westerveld T., Hiemstra D., Blok H.E., and de Vries A.P. Vtijah scratches INEX 2005: Vague element selection, image search, overlap, and relevance feedback. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 72–87.
7. O'Keefe R.A. If INEX is the answer, what is the question? In Proc. 3rd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 54–59.
8. Sauvagnat K., Hlaoua L., and Boughanem M. Xfirm at INEX 2005: ad-hoc and relevance feedback tracks. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 88–103.
9. Theobald M., Schenkel R., and Weikum G. Topx and xxl at INEX 2005. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2006, pp. 282–295.
10. Trotman A. Wanted: element retrieval users. In Proc. INEX 2005 Workshop on Element Retrieval Methodology, 2005, pp. 63–69.
11. Trotman A. and Lalmas M. Strict and vague interpretation of XML-retrieval queries. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 709–710.
12. Trotman A. and Lalmas M. Why structural hints in queries do not help XML retrieval. In Proc. 32nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2006, pp. 711–712.
13. Woodley A., Geva S., and Edwards S.L. Comparing XML-IR query formation interfaces. Australian J. Intelligent Inf. Proc. Syst., 9(2):64–71, 2007.
14. van Zwol R. B^3-sdr and effective use of structural hints. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 146–160.
15. van Zwol R., Baas J., van Oostendorp H., and Wiering F. Bricks: the building blocks to tackle query formulation in structured document retrieval. In Proc. 28th European Conf. on IR Research, 2006, pp. 314–325.

Processor Cache

Peter Boncz
CWI, Amsterdam, The Netherlands

Synonyms

Data cache; Instruction cache; CPU cache; L1 cache; L2 cache; L3 cache; Translation Lookaside Buffer (TLB)

Definition

To hide the high latencies of DRAM access, modern computer architecture now features a memory hierarchy that besides DRAM also includes SRAM cache memories, typically located on the CPU chip. Memory access first check these caches, which takes only a few cycles. Only if the needed data is not found, an expensive memory access is needed.

Key Points

CPU caches are SRAM memories located on the CPU chip, intended to hide the high latency of accessing off-chip DRAM memory. Caches are organized in cache lines (typically 64 bytes). In a fully-associative cache, each memory line can be stored in any location of the cache. To make checking the cache fast, however, CPU caches tend to have limited associativity, such that storage of a particular cache line is possible in only 2 or 4 locations. Thus only 2 or 4 locations need to be checked during lookup (these are called 2- resp. 4-way associative caches). The cache hit ratio is determined by the spatial and temporal locality of the memory accesses generated by the running program(s).

Cache misses can either be compulsory misses (getting the cache lines of all used memory once), capacity misses (caused by the cache being too small to keep all multiply used lines in cache), or conflict misses (due to the limited associativity of the cache).

Most modern CPUs have at least three independent caches: an instruction cache to speed up executable instruction fetch, a data cache to speed up data fetch and store, and a Translation Lookaside Buffer (TLB) used to speed up virtual-to-physical address translation for both executable instructions and data. The TLB is not organized in cache lines, it simply holds pairs of (virtual, logical) page mappings, typically a fairly limited amount (e.g., 64). In practice, this mean that algorithms that repeatedly touch memory in more than 64 pages (whose size is often 4 KB) shortly after each other, run into TLB thrashing. This problem can sometimes be mitigated by setting a large virtual memory page size, or by using special large OS pages (sometimes supported in the CPU with a separate, smaller, TLB for large pages).

Another issue is the tradeoff between latency and hit rate. Larger caches have better hit rates but longer latency. To address this tradeoff, many computers use multiple levels of cache, with small fast caches backed

up by larger slower caches. Multi-level caches generally operate by checking the smallest Level 1 (L1) cache first; if it hits, the processor proceeds at high speed. If the smaller cache misses, the next larger cache (L2) is checked, and so on, before external memory is checked. As the latency difference between main memory and the fastest cache has become larger, some processors have begun to utilize as many as three levels of on-chip cache.

For multi-CPU and multi-core systems, the fact that some of the higher levels of cache are not shared, yet provide coherent access to shared memory, causes additional cache-coherency inter-core communication to invalid stale copies of cache lines on other cores when one core modifies it. In multi-core CPUs, an important issue is which cache level is shared among all cores – this cache level is on the one hand a potential hot-spot for cache conflicts, on the other hand provides an opportunity for very fast inter-core data exchange.

In case of sequential data processing, the memory controller or memory chipset in modern computers often detect this access pattern and start requesting the subsequent cache lines in advance. This is called hardware prefetching. Prefetching effectively allows to hide compulsory cache misses. Without prefetching, the effective memory bandwidth would equate cache line size divided my memory latency (e.g., 64/50 ns = 1.2 GB/s). Thanks to hardware prefetching, modern computer architectures reach four times that on sequential access. Modern CPUs also offer explicit prefetching instructions, which a software writer can exploit to perform (non-sequential) memory accesses in advance, hiding their latency. In database systems, such software prefetching has successfully been used in making hash--table lookup faster (e.g., in hash-join and hash-aggregation).

In database systems, a series of cache-conscious data storage layouts (e.g., DSM and PAX) have been proposed to improve cache line usage. Also, a number of cache-conscious query processing algorithms, such as cache-partitioned hash join and hash-join using memory prefetching, have been studied. In the area of data structures and theoretical computer science, there has recently been interest in cache-oblivious algorithms, that regardless the exact parameters of the memory hierarchy (number of levels, cache size, cache line sizes and latencies) perform well.

Cross-references

- ▶ Architecture-Conscious Database System
- ▶ Cache-Conscious Algorithms
- ▶ Disk
- ▶ DSM
- ▶ Main Memory
- ▶ Main Memory DBMS
- ▶ PAX Storage Layouts

Production-based Approach to Media Analysis

- ▶ Computational Media Aesthetics

Projected Clustering

- ▶ Subspace Clustering Techniques

Projection

Cristina Sirangelo
University of Edinburgh, Edinburgh, UK

Definition

Given a relation instance R over set of attributes U, and given a subset X of U, the projection of R on X – denoted by $\pi_X(R)$ – is defined as a relation over set of attributes X whose tuples are the restriction of tuples of R to attributes X. That is $t \in \pi_X(R)$ if and only if $t = t'(X)$ for some tuple t' of R (here $t'(X)$ denotes the restriction of t' to attributes X).

Key Points

The projection is one of the basic operators of the relational algebra. It operates by "restricting" the input relation to some of its columns.

The arity of the output relation is bounded by the arity of the input relation. Moreover the number of tuples in $\pi_X(R)$ is bounded by the number of tuples in R. In particular, the size of $\pi_X(R)$ can be strictly smaller than the size of R since different tuples of R may have the same values on attributes X.

As an example, consider a relation *Goods* over attributes (*code, price, quantity*), containing tuples

{(001, 5.00, 10), (002, 5.00, 10), (003, 25.00, 3)}. Then $\pi_{price,quantity}(Goods)$ is a relation over attributes (*price*, *quantity*) with tuples {(5.00, 10), (25.00, 3)}.

In the case that attribute names are not present in the relation schema, the projection is specified by an expression of the form $\pi_{i_1,...,i_n}(R)$. Here $i_1,...,i_n$ is a sequence of positive integers – where each i_j is bounded by the arity of R – identifying attributes of R. The output is a relation of arity n with tuples $(t(i_1),..., t(i_n))$, for each tuple t in R. In this case the arity n of the output relation can be possibly larger than the arity of R, since integers $i_1,...,i_n$ need not be distinct.

Cross-references

► Relation
► Relational Algebra

Projection Index

► Bitmap-based Index Structures for Multidimensional Data

Propagation-based Structured Text Retrieval

Karen Pinel-Sauvagnat
IRIT-SIG, Toulouse Cedex, France

Synonyms

Relevance propagation; Score propagation

Definition

Evaluate the relevance score of text components. Approaches using propagation view the logical structure of a structured document as a tree whose nodes are components of the document and whose edges represent the relationships between the connected nodes. A document component can be either a leaf or an inner node. Leaf nodes are document components that correspond to the last elements of hierarchical relationship chains and that contain raw data (textual information). With propagation, relevance scores are first calculated for leaf components. They are then propagated upwards in the document tree structure to calculate relevance scores for the inner components.

Historical Background

With appropriate query languages, users may want to exploit the structure of structured text documents to perform fine-grained and flexible retrieval. Instead of treating documents as atomic units, structured text retrieval systems aim thus at retrieving document components that answer a given information need in the most *specific* way. Classical information retrieval (IR) models (e.g., the vector space model, the probabilistic model, language models) have been extended in order to take into account the structural dimension of documents. IR research has shown that document term weighting is a crucial concept for effective retrieval, thus classical formalisms were adapted with additional weighting parameters: component type, number of descendant components, frequency of the component type, to name a few. However, most of these adaptations do not use (or make little use of) the tree representation of structured documents to identify the most specific components.

The idea behind relevance propagation method is to follow the way *relevance* changes in a document tree to estimate the relevance of the document components. Indeed *relevance* in structured text documents has been expressed in terms of *specificity* and exhaustivity: the specificity (its coverage of the topic and nothing else) of components in the tree typically decreases as one moves up the tree, and when a component has multiple relevant descendants, its exhaustivity (coverage of the topic) usually increases compared to that of each of the descendant elements.

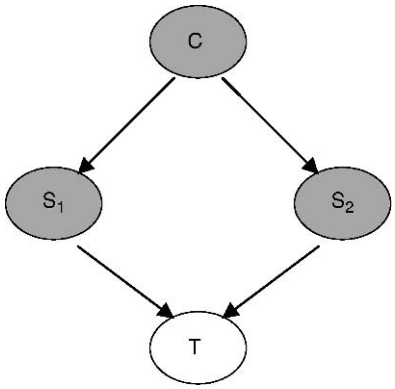
In propagation methods, relevance scores are thus first computed at leaf level, and then propagated up the document tree: to allow the identification of the most specific components, the relevance score of leaf components should be somehow decreased while being propagated upwards in the document tree, and to allow the identification of the most exhaustive components, relevance scores may be aggregated (a parent score should be evaluated using its children scores). A naive solution would be to just sum the relevance scores of each inner component relevant children. However, this would ultimately result in root components being returned at top ranks although they may not be the most specific components for the given information need.

Few relevance propagation approaches were proposed before 2002 (when *INEX*, the evaluation Initiative for XML Retrieval, was set-up). One can however cite the method presented in [5] using inference nets. The retrieval process is applied to SGML documents but can be extended to any type of structured documents. The basic retrieval strategy is to calculate the degree to which a component at any level of the document hierarchy satisfies the query by considering what components are contained in that component. This strategy makes it possible to systematically calculate the expected relevance of a component at any level, taking into account its relationship with other components in the hierarchy. Consider the following net made of two section components S_1 and S_2 that are the parent node of a term node T and the children nodes of the component C (see Fig. 1).

The retrieval process is performed in a bottom-up fashion, because of the way documents are represented in the inference net: no components other than leaf components contain actual text, and the retrieval process must start from leaf components whose text contains a query term.

The degree of belief of T given the network topology is computed with a simplified formula as follows (the simplification comes from considering only positive events):

$$\begin{aligned} B(T|C) &= P(T|S_1) \times P(S_1) + P(T|S_2) \times P(S_2) \\ &= P(T|S_1) \times P(S_1|C) \times P(C) + P(T|S_2) \\ &\quad \times P(S_1|C) \times P(C). \end{aligned} \tag{1}$$



Propagation-based Structured Text Retrieval. Figure 1. A simple network.

Children components S_i and their parent C are represented as $S_i = <s_{i1}, s_{i2},...,s_{in}>$ and $C = <c_1, c_2,...,c_n>$ respectively, where n is the number of index terms and s_{ij} and c_i are calculated using standard term frequency (TF) and inverse document frequency (IDF) statistics. The probability $P(S_i|C)$ of observing S_i given C (or the degree of belief that C supports S_i) is calculated as a similarity between the two vectors:

$$P(S_i|C) = \lambda_i \times (S_i \cdot C), \tag{2}$$

where λ_i is a weight associated to the component type, representing its overall importance relative to other types of components sharing the same parent. This computation incorporates both the content and the type of components.

The probability $P(T|S_i)$ of observing a term T given a component S_i (or the degree of belief that S_i supports T) is estimated with:

$$P(T|S_i) \cong IDF_T \times TF_{i,S_i}. \tag{3}$$

$P(C)$ is the probability of observing C assuming that C is the root node (i.e., document) in the inference net. In the implementation, it is set to 1. This approach was however not evaluated, since no suitable test collection was available.

Other approaches presented in the rest of the entry were developed and evaluated in the context of the *INEX* evaluation campaign, which is concerned with the evaluation of XML retrieval. In this case, document components correspond to XML elements.

Foundations

As all IR approaches, propagation-based approaches can be caracterized in terms of indexing and scoring methods (here both for leaf and inner elements).

Indexing

Approaches using relevance propagation consider *indexing units* as disjoint units: the text of each element is the union of one or more of its disjoint parts. Thus, as textual information is only present in leaf elements, inverted index only concern leaf elements. Propagation accounts then for the fact that the text in a given leaf element is also contained in its ancestors. The document structure is generally stored in a separate index and used to build the document trees.

Relevance Scores Evaluation for Content-Only Queries In the rest of the entry, the following notations are used. Let q be a query composed of k terms $t_1,...t_k$. In the document tree, *le* is a leaf element and *e* an inner element having n children. $RSV(q, le)$ is the relevance score of the leaf element *le* with respect to query q and $RSV(q, e)$ is the final score of element *e* with respect to query q.

Scoring Leaf Elements To evaluate leaf element scores ($RSV(q, le)$), approaches found in the literature have used the following parameters:

1. Frequency of term t_i in query q or leaf element *le*
2. Frequency of term t_i in the whole collection
3. Number of leaf elements containing t_i
4. Length of leaf element
5. Average length of leaf elements
6. Inverse element frequency *ief*, which is similar to *idf* (inverse document frequency) but that takes into account the collection of leaf elements instead of the collection of documents.

In [1], the weighting scheme used to compute the relevance score of leaf elements also uses the cross-structural importance of t_i relative to *le* and q, which allows to increase or decrease the importance of a term depending on its location in the query (some terms may be more important than others in a content-and-structure query). In [3], the frequency of terms in the leaf elements and in the whole collection are used with a parameter scaling up the score of elements having multiple query terms. The formula penalizes elements with frequently occurring query terms (frequent in the collection), and rewards elements with more unique query terms within a result element. One can also cite the leaf elements weighting scheme presented in [9], where term frequency and inverse element frequency *ief* are applied together with *idf*. This allows to take into account the importance of terms in both the collection of leaf elements and the collection of documents.

Propagating Relevance Scores Once the relevance score of the leaf elements have been calculated, they are used to evaluate the relevance score of the inner elements. The main issue here is how to combine these scores. As already said, a naive solution that simply sums the relevance sores of leaf elements will result in root elements being ranked at the top of the result lists, although they are likely to not constitute the most specific elements to the query.

In [1] the relevance of inner elements is for instance evaluated using the maximum of their leaf element scores. Other approaches use a weighted sum of leaf or children element scores, and make some assumptions related to the document tree structure. They are described below.

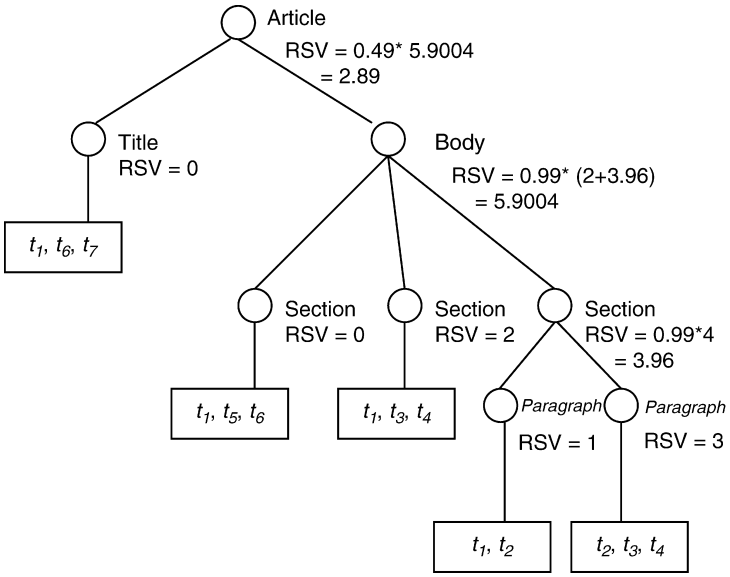
For example, in the GPX approach [3], a heuristically derived formula is proposed to evaluate the scores of inner elements that accounts for specificity and exhaustivity:

$$\mathrm{RSV}(q, e) = D(n) \sum_{l=1}^{n} \mathrm{RSV}(q, l), \qquad (4)$$

where n is the number of children elements, $D(n) = 0.49$ if $n = 1$, 0.99 otherwise, and $RSV(q, l)$ is the relevance score of the lth child element.

The value of the decay factor D depends on the *number of relevant children* that the inner element has. If the element has one relevant child then the decay constant is 0.49. An element with only one relevant child will be ranked lower than its child. If the element has multiple relevant children the decay factor is 0.99. An element with many relevant children will be ranked higher than its descendants. Thus, a section with a single relevant paragraph would be judged less relevant than the paragraph itself, but a section with several relevant paragraphs will be ranked higher than any of the paragraphs.

"t_2 t_3 t_4". The method is illustrated in Fig. 2, with the content-only query "t_2 t_3 t_4". To simplify, the weight of a term in a leaf element is equal to 1 if it is a query term and 0 otherwise. For example the /article[1]/body[1]/section[3]/paragraph[2] element that contains three query terms has thus a score of 3. Its parent (/article[1]/body[1]/section[3]) contains two relevant children with scores 1 and 3. Its score is then calculated with a decay factor equal to 0.99 as follows: $0.99 \times (1 + 3)$. To evaluate the score of the root element (which only contains one relevant child), the decay factor used is 0.49, and the root element has consequently a lower score than its child /article[1]/body[1]. As a result of the propagation, the /article[1]/body[1] element has the highest score and will be ranked first by the GPX system.



Propagation-based Structured Text Retrieval. Figure 2. Relevance propagation according to [3].

The *distance between an element and its descendant leaf elements* has also been used as a weight in the weighted sum that calculates the relevance score of inner elements. Terms that occur close to the root of a given subtree are more significant to the root element that ones at deeper levels of the subtree. It seems therefore that the greater the distance of a element from its ancestor, the less it should contribute to the relevance of its ancestor. This can be modeled with the $d(x, y)$ parameter, which is the distance between elements x and y in the document tree, i.e., the number of arcs joining x and y.

In [4], the relevance score of an inner element takes into account this distance and also the distance separating the root element and leaf elements. The latter is used as a normalization factor. The relevance score of an inner element e is calculated as follows:

$$\mathrm{RSV}(q,e) = \sum_{le_j \in L_e} \left(1 - 2\lambda\lambda \frac{d(e, le_j)}{d(e, le_j) + d(root, le_j)}\right)^2 \mathrm{RSV}(q, le_j). \quad (5)$$

λ is a constant coefficient ≥ 0, le_j are leaf elements being descendant of e, and L_e is the set of leaf elements being descendant of e. This process tends to consider an element having a relevant descendant element less relevant than the descendant element itself, which is comparable to the approach followed in [3], as above described.

Finally, in [9], the distance between elements and the number of relevant descendant leaf elements are used together with an additional parameter, β, in the weighted sum. The β factor captures the assumption that small elements may be used by authors to highlight important information (*title* elements, *bold* elements, ...). *Small elements can therefore give important indications on the relevance of their ancestors* and their importance should be increased during propagation. The relevance value of an element e is thus computed according to the following formula:

$$\mathrm{RSV}(q,e) = |L_e^r| . \sum_{le_j \in L_e} \alpha^{d(e, le_j)-1} \times \beta(le_j) \times \mathrm{RSV}(q, le_j), \quad (6)$$

where α is a constant coefficient $\in]0,1]$ used to tune the importance of the distance $d(e, le_j)$ parameter and $|L_e^r|$ is the number of leaf elements being descendant of e and having a non-zero relevance value. $\beta(le_j)$ is experimentally fixed and allows to increase the role of elements smaller than the average leaf element size in the propagation function.

Sauvagnat et al. (2005) [9] also propose a backward propagation after the first propagation, in order to account for the whole document relevance (and thus for the element context) in the calculation of the relevance score of inner elements.

Content-and-Structure Queries Processing

In most of the approaches using relevance propagation, results elements are simply filtered to satisfy the structural constraints of *content-and-structure queries* [3,4]. The approach described in [8] however uses relevance propagation to process structural constraints. Queries may have several constraints, and each constraint is composed of both a content and a structure condition, one of them indicating which type of elements should be returned (target elements). For each constraint, propagation starts from leaf nodes answering the content condition and goes until an element that matchs the structure constraint is found. The score of result elements of each constraint is then propagated again to elements belonging to the set of targeted structures.

Key Applications

The access to structured text documents such as SGML or XML documents is the main application of the relevance propagation approach described in this entry. Propagation can also be used to access HTML documents in the context of web retrieval, for example to determine among a set of related web pages, which one corresponds to the best entry in the set. Moreover, propagation can also be applied to distributed IR, as presented in [2].

Future Directions

Propagation methods presented in this entry are relatively independent of the DTDs of collections, since most of them do not use the type of elements to estimate relevance. However, propagation cannot be done in the same way on small document collections and large document collections. Methods should be adapted to process large document collections and efficiency issues that follow.

Relevance propagation may evolve in the future with the use of another source of evidence to calculate inner element relevance score, for example, link information. Indeed, web approaches using relevance propagation have also used links to find relevant web pages (a relevant page may be linked to other relevant pages) [7]. The same approaches could be used for structured text retrieval.

Experimental Results

INEX evaluation campaign. For example, the approach in [3] was ranked in the top 5 for the focused retrieval task (which aims at targeting the appropriate level of granularity of relevant content that should be returned to the user for a given topic) and comparable results were achieved by the approach described in [8] using content-and-structure queries.

Cross-references

- ▶ Aggregation-Based Structured Text Retrieval
- ▶ INEX
- ▶ On Enlve
- ▶ Relevance
- ▶ Specificity
- ▶ Term Statistics for Structured Text Retrieval
- ▶ XML Retrieval

Recommended Reading

1. Anh V.N. and Moffat A. Compression and an IR approach to XML Retrieval. In Proc. 1st Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2002.
2. Baumgarten C. A probabilistic model for distributed information retrieval. In Proc. 20th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1997, pp. 258–266.
3. Geva S. GPX – Gardens Point XML IR at INEX 2005. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 240–253.
4. Hubert G. XML retrieval based on direct contribution of query components. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 172–186.
5. 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.
6. Ogilvie P. and Callan J. Parameter estimation for a simple hierarchical generative model from XML retrieval. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 211–224.
7. Qin T., Liu T.-Y., Zhang X.-D., Chen Z., and Ma W.-Y. A study of relevance propagation for web search. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 408–415.
8. Sauvagnat K., Boughanem M., and Chrisment C. Answering content-and-structure-based queries on XML documents using relevance propagation. Inf. Syst., 31:621–635, 2006.
9. Sauvagnat K., Hlaoua L., and Boughanem M. XFIRM at INEX 2005: adhoc and relevance feedback tracks. In Proc. 4th Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2005, pp. 88–103.

Protein Sequence

- ▶ Biological Sequence

Protein-Protein Interaction Networks

► Biological Networks

Provenance

Wang-Chiew Tan
University of California-Santa Cruz, Santa Cruz, CA, USA

Synonyms

Lineage; Origin; Source; History; Pedigree

Definition

Let t be a data element in the result of a query Q applied to a dataset D. A data element may be a tuple in the relational model, or a subtree in the semi-structured model. The *provenance* of t is the set of all proofs for t according to Q and D. A proof for t according to Q and D is a set D' of data elements in D so that t is in the result of applying Q on D'. In some cases, a proof also details the process by which t is derived from Q and D'.

Most work on provenance in databases focused on finding data elements of D that witness the existence of t in the result, as well as which data elements of D is t copied from. In scientific workflows, the provenance of a result is typically a detailed description of the entire workflow used to arrive at the result.

Key Points

The need to understand and manage the provenance of data arises in almost every application. For example, it is natural to ask for the provenance of data seen on the Web. In scientific experiments, the provenance of results generated by the experiments constitutes to the proof of correctness directly or indirectly, and is typically regarded to be as important as the result itself.

There are generally two types of provenance: data provenance and workflow provenance [1,3].

Data provenance is an account of the derivation of a piece of data in a dataset that is typically the result of a database operation on an input database. There are two approaches for computing data provenance [2]: annotation-based versus non annotation-based approaches. In annotation-based approaches, provenance is captured by propagating annotations (i.e., extra information) from input to output along database transformations. Hence, the provenance of an output data can typically be determined by analyzing the associated annotations. In contrast, non annotation-based approaches do not carry along extra information. Instead, the provenance of an output data is in the determined by analyzing the database transformation, as well as the input and output databases.

Workflow provenance refers to the record of the history of the derivation of some dataset in a scientific workflow. The amount of information recorded for workflow provenance varies, depending on application needs. For example, workflow provenance of a scientific result may include details about the type and model of external devices used, as well as the versions of softwares used for deriving the result.

Cross-references

► Data Provenance
► Provenance
► Provenance in Experimental Data Management
► Provenance in Scientific Databases
► Storage Security

Recommended Reading

1. Bose R. and Frew J. Lineage retrieval for scientific data processing: a survey. ACM Comput. Surv., 37(1):1–28, 2005.
2. Buneman P. and Tan W.-C. Provenance in databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1171–1173.
3. Simmhan Y., Plale B., and Gannon D. A survey of data provenance in E-Science. ACM SIGMOD Rec., 34:31–36, 2005.

Provenance Metadata

► Data Provenance

Provenance in Scientific Databases

Sarah Cohen-Boulakia[1], Wang-Chiew Tan[2]
[1]University of Pennsylvania, Philadelphia, PA, USA
[2]University of California-Santa Cruz, Santa Cruz, CA, USA

Synonyms

Lineage; Origin; Source; History; Pedigree

Definition

Scientific databases contain data which may have been produced as answer to a query posed over other resources, or generated by in-silico experiments (or scientific workflow) involving various softwares, or manually curated by domain experts based on analysis of several other resources. The provenance of a piece of data in scientific databases typically includes information of where this piece of data originates from, as well as details of the scientific process (e.g., parameters used in the experiments, software versions etc.) by which it arrived in the scientific database.

Historical Background

Provenance of scientific databases has been studied in two granularities: *workflow provenance* and *data provenance*.

Workflow provenance (or *coarse-grained provenance*) refers to the record of the history (or workflow) of the derivation of some dataset in a scientific workflow [5,13,14]. The amount of information recorded for workflow provenance varies, depending on application needs. For example, in some cases, a workflow provenance may record the type and model of external devices used, such as sensors, cameras, or other data collecting equipments, as well as the associated versions of software used for processing data; in other cases, these information may be deemed unnecessary.

More recently, workflow systems developed within the scientific community to conduct and manage experiments have started recording information about the processes used to derive intermediate and final data objects from raw data. Survey papers dedicated to workflow provenance approaches have been proposed [5,14] and "provenance challenges" [13] have been held to encourage system designers to learn about the capabilities and expressiveness of each others' systems and work towards interoperable solutions.

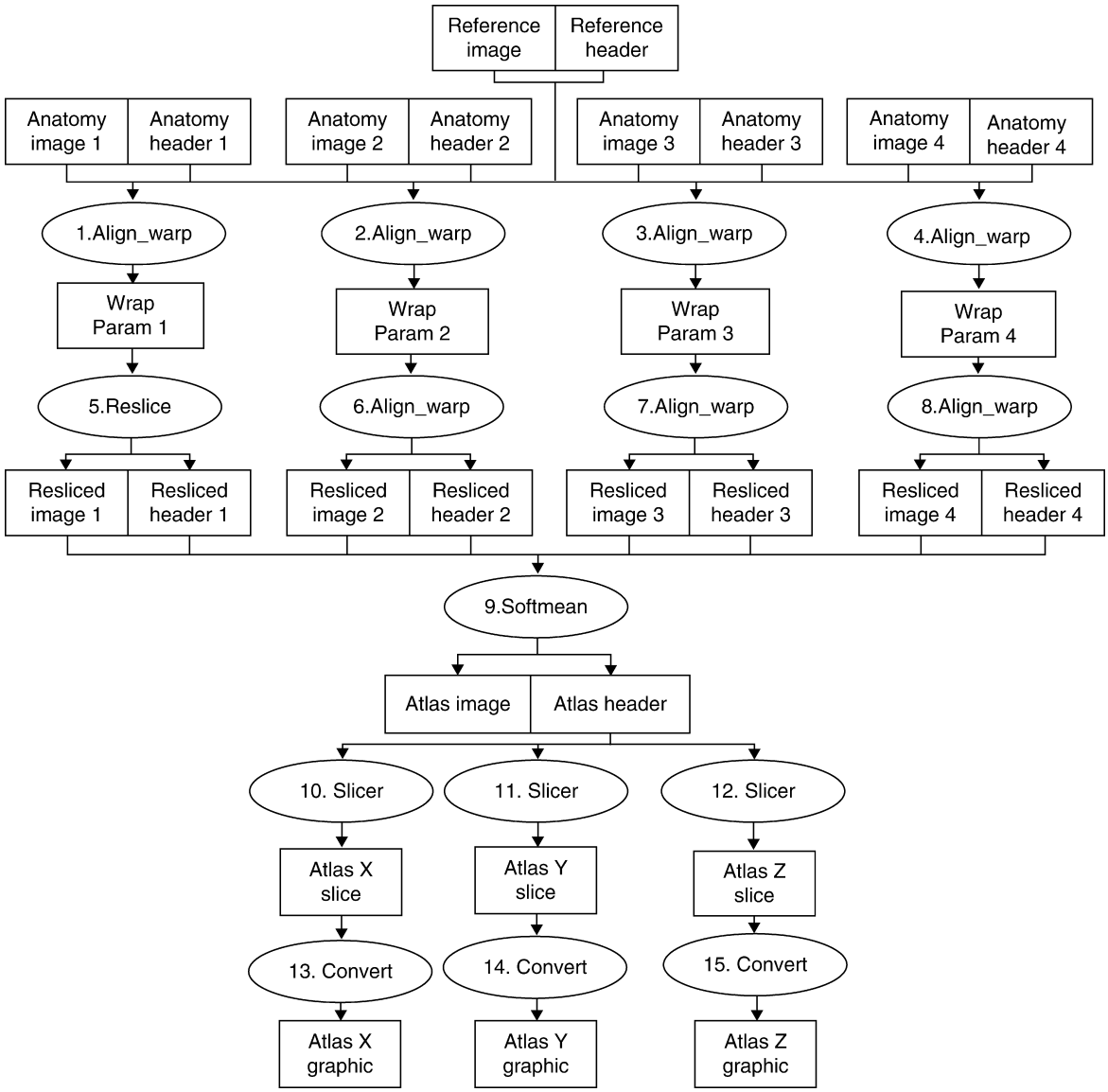
Data provenance (or *fine-grained provenance*) is an account of the derivation of a piece of data in a dataset that is typically the result of executing a database query against a source database [9]. This type of provenance determines the parts of source data that were used to generate a piece of data in the resulting dataset. Typically, data provenance is obtained by carefully reasoning about the algebraic form of the database query and the underlying data model of the source and resulting databases. In contrast, transformations occurring in scientific workflows are often external processes (e.g., perl scripts), and the log files of scientific workflows provide only object identifiers to pieces of data involved in the transformation at best. Often, such external processes and data involved do not possess good properties for detailed analysis. Hence, a fine-grained analysis of an external process is not always possible and the provenance recorded for such transformation is usually more coarse-grained.

Foundations

Workflow Provenance: The notion of workflow provenance is illustrated using the workflow of the first provenance challenge, which aimed to establish an understanding of the capabilities of available provenance-related systems. This simple workflow is inspired from a real experiment in the area of functional Magnetic Resonance Imaging (fMRI) and formed the basis of the challenge. It is represented in Fig. 1 as a graph where oval nodes represent steps and rectangle nodes represent kind of data exchanged between them. Typical provenance queries ask for the history of some data item, e.g., "What caused Atlas X Graphic (one final output) to be as it is?". Depending on whether the complete history of the data is asked or only the previous step and its inputs, queries are qualified as *deep* (or *recursive*) or *immediate*. For example, the immediate provenance of *Atlas X Graphic* is given by the step *13.convert* and its input, *Atlas X slice*, while its deep provenance is given by all the data and steps used to compute the step *9.softmean* and *Atlas Image, Atlas Header, 10.slicer, Atlas X slice*, and *13.convert*. Other provenance queries may include *annotation queries* finding data annotated with some specific metadata, such as "Find all invocations of procedure align_warp using a twelfth order nonlinear 1365 parameter model?" (metadata on the procedure used should be recorded) and "Find the outputs of align_warp where the inputs are annotated with center = UChicago" (metadata on the data used should be recorded).

Various workflow provenance approaches have been designed and applied to very diverse scientific domains (e.g., biology, ecology, astronomy, meteorology). As workflow provenance is by nature associated with actual scientific experiments, most of these approaches have been implemented into prototypes or systems that have typically participated in such challenges [13].

Approaches differ in various aspects. In particular, provenance can be tracked at various degrees of granularity: from OS or instrument level to input/output data and metadata associated to them. The graphs formed by the workflow provenance information are



Provenance in Scientific Databases. Figure 1. Workflow of the first provenance challenge.

also represented using various data models, from relational to tree-based data models (e.g., XML, such as in the Kepler and ES3 projects) or graph-based data models (e.g., Web semantic approaches based on RDF, such as in the myGrid or Wings/Pegasus projects). As a consequence, a plethora of languages have been used to query workflow provenance information including SQL, XQuery, and SPARQL while several dedicated graph query languages have been designed.

More precisely, Foster et al. [13] tackle with the problem of reproducing experiments by providing a *virtual data system* approach offering the ability to not only track the data consumed and produced during computations but also rederive deleted intermediate results of an experiment (virtual data).

Callahan et al. [13] focus on the problem of *workflow evolution*: *VisTrails* provides techniques to compute workflow differences and similarities.

In an orthogonal way, Miles et al. define an *open architecture for provenance* [13] designed to be domain and technology independent. In this architecture, *process documentations* describe what actually occurred at execution time and are created and stored by provenance-aware applications.

Finally, Biton et al. provide abstraction mechanisms to help users face the overwhelming amount of provenance information in workflow environments, offering the possibility of focusing on the most relevant provenance information. The technique pursued in *ZOOM*UserViews* is that of "user views" [3]: Since bioinformatics tasks may themselves be complex sub-workflows, a user view determines what level of sub-workflow the user can see, and thus what data and steps are visible in provenance queries. Algorithms to compute relevant user views have been proposed in [4].

Data Provenance: Here, biological databases are used as example scientific databases for illustrating data provenance. The relations and SQL query are shown below.

```
SELECT swissprot, pir, s.desc
FROM SWISS-PROT s, PIR p, Mapping
     Table m
WHERE s.id = m.swissprot AND p.id = m.
      pir
```

The result of executing the SQL query consists of two tuples (a231, p445, AB) and (w872, p267, CD). Work on data provenance [7,11] has provided explanations for why a tuple, such as (a231, p445, AB), is in the query result. The reason is because the first tuples in SWISS-PROT, PIR and Mapping-Table respectively, joined according to the query to produce the output tuple. This type of provenance is termed *why-provenance* in [7]. In contrast, the *where-provenance* of "AB" in the output tuple (i.e., where "AB" is copied from) is the desc attribute of the first tuple in SWISS-PROT. In particular, no other source tuples are involved in the explanation of where-provenance. Subsequent work by Green et al. [12] is also able to explain *how* a tuple is derived in the result of a query.

Another line of work captures provenance by propagating annotations of source data to the output, based on provenance, along query transformations. Wang and Madnick [15] first articulated the idea of using propagated annotations to analyze source attribution. Subsequently, Buneman et al. [8] studied the *annotation placement problem* using a variation

SWISS-PROT

id	desc
a231	AB
w872	CD
u812	DD

PIR

id	desc
p445	AB
p267	CD
p547	ED

Mapping-Table

mid	swissprot	pir
1	a231	p445
2	w872	p267

of the propagation scheme of [15]. The propagation scheme of [8,15] is essentially based on where data is copied from. This scheme forms the *default* propagation scheme of DBNotes [2]. A serious drawback of the default scheme is that two equivalent SQL queries (select-project-join-union queries) may not propagate annotations in the same way. In DBNotes, the authors proposed an alternative *default-all* scheme to overcome this limitation. In the default-all propagation scheme, equivalent queries always propagate annotations in the same way. Additionally, DBNotes also support the *custom* propagation scheme, where one is allowed to customize a propagation scheme as desired. So far, all systems [2,8,15] only allow annotations on attribute values. Subsequent research efforts have proposed techniques to relax this restriction.

Most database and workflow techniques provide little help for managing provenance in curated databases. This is because curated databases are, by definition, not constructed from the result of executing a query but rather, created manually by scientists through the analysis of information from several sources. In [6], the authors proposed a copy-and-paste model that captures a scientist's manual curation efforts as provenance-aware transactions. These transactions are logged and various hierarchical compression techniques were proposed and validated experimentally.

Key Applications

There are many application domains that can benefit from a provenance system in science. Several uses of

provenance information are considered here, as described in the literature [5,13,14].

Informational: Interpreting and understanding the result of a scientific experiment necessitates to know the provenance or context in which the data has been produced.

Data Quality: By providing the source data and transformations, provenance may help to estimate data quality and data reliability.

Error Detection: Provenance can be used to detect errors in data generation.

Re-use: Reproducing a local experiment or an experiment described in a research paper may be possible if precise provenance information is provided.

Attribution: The copyright and ownership of data can be determined using provenance information. It enables its citation, and may determine liability in case of erroneous data.

Probabilistic Databases: Provenance has been used to compute the confidences in the result of a probabilistic query correctly, as well as to correctly reason about the set of possible instances in the result of the query [1].

Data Sharing and Data Integration: Provenance has also been used to describe trust policies in the data sharing system Orchestra and to prioritize updates, as well as to understand and debug a data exchange or data integration specification [10].

URL to Code

Chimera Project: http://www.ci.uchicago.edu/wiki/bin/view/VDS/VDSWeb/WebMain

DBNotes Project: http://www.soe.ucsc.edu/~wctan/Projects/dbnotes

ES3 Project: http://eil.bren.ucsb.edu

GriPhyN Virtual Data System Project: http://www.ci.uchicago.edu/wiki/bin/view/VDS/VDSWeb/WebMain

Kepler Project: http://kepler-project.org

myGrid Project: http://www.mygrid.org.uk

Orchestra Project: http://www.cis.upenn.edu/~zives/orchestra

Pasoa Project: http://users.ecs.soton.ac.uk/lavm/projects/pasoa.html

Provenance Challenges: http://twiki.ipaw.info/bin/view/Challenge

SPIDER Project: http://www.soe.ucsc.edu/wctan/Projects/debugger/index.html

VisTrails Project: http://vistrails.sci.utah.edu/index.php/Main_Page

Wings/Pegasus Project: http://www.isi.edu/ikcap/wings

ZOOM*UserViews Project: http://zoomuserviews.db.cis.upenn.edu

Cross-references

- ▶ Data Provenance
- ▶ Provenance
- ▶ Provenance in Experimental Data Management
- ▶ Scientific Databases
- ▶ Scientific workflows

Recommended Reading

1. Benjelloun O., Sarma A.D., Halevy A.Y., and Widom J. ULDBs: databases with uncertainty and lineage. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 953–964.
2. Bhagwat D., Chiticariu L., Tan W.C., and Vijayvargiya G. An annotation management system for relational databases. VLDB J., 14(4):373–396, 2005.
3. Biton O., Cohen-Boulakia S., and Davidson S. Zoom* UserViews: querying relevant provenance in workflow systems. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 1366–1369.
4. Biton O., Cohen-Boulakia S., Davidson S., and Hara C.S. Querying and managing provenance through user views in scientific workflows. In Proc. 24th Int. Conf. on Data Engineering, 2008.
5. Bose R. and Frew J. Lineage retrieval for scientific data processing: a survey. ACM Comput. Surv., 37(1):1–28, 2005.
6. Buneman P., Chapman A., and Cheney J. Provenance management in curated databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 539–550.
7. Buneman P., Khanna S., and Tan W.C. Why and where: a characterization of data provenance. In Proc. 8th Int. Conf. on Database Theory, 2001, pp. 316–330.
8. 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.
9. Buneman P. and Tan W.C. Provenance in databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1171–1173.
10. Chiticariu L. and Tan W.C. Debugging schema mappings with routes. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 79–90.
11. Cui Y., Widom J., and Wiener J.L. Tracing the lineage of view data in a warehousing environment. ACM Trans. Database Syst., 25(2):179–227, 2000.
12. Green T.J., Karvounarakis G., and Tannen V. Provenance semirings. In Proc. 26th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2007, pp. 31–40.
13. Moreau L., Ludäscher B., Altintas I., Barga R.S., Bowers S., Callahan S., Chin G., Clifford B., Cohen S., Cohen-Boulakia S.,

Davidson S., Deelman E., Digiampietri L., Foster I., Freire J., Frew J., Futrelle J., Gibson T., Gil Y., Goble C., Golbeck J., Groth P., Holland D.A., Jiang S., Kim J., Koop D., Krenek A., McPhillips T., Mehta G., Miles S., Metzger D., Munroe S., Myers J., Plale B., Podhorszki N., Ratnakar V., Santos E., Scheidegger C., Schuchardt K., Seltzer M., Simmhan Y.L., Silva C., Slaughter P., Stephan E., Stevens R., Turi D., Vo H., Wilde M., Zhao J., and Zhao Y. The first provenance challenge. Concurrency Comput. Pract. Exp., 20(5):409–418, 2007, Special issue on the First Provenance Challenge.

14. Simmhan Y., Plale B., and Gannon D. A survey of data provenance in e-science. ACM SIGMOD Rec., 34:31–36, 2005.
15. Wang Y.R. and Madnick S.E. A polygen model for heterogeneous database systems: the source tagging perspective. In Proc. 16th Int. Conf. on Very Large Data Bases, 1990, pp. 519–538.

Proximity

► Similarity and Ranking Operations

PRP

► Probability Ranking Principle

p-Sensitive k-Anonymity

► k-Anonymity

Pseudonymity

Simone Fischer-Hübner
Karlstad University, Karlstad, Sweden

Synonyms

Nymity

Definition

The term pseudonymous originates from the Greek word "*pseudonymos*" meaning "having a false name."

A pseudonym is an identifier of a subject other than one of the subject's real names, and pseudonymity is the use of pseudonyms as identifiers. Sender pseudonymity is defined as the sender being pseudonymous, recipient pseudonymity as the recipient being pseudonymous [2].

Key Points

Pseudonymity resembles anonymity as both concepts aim at protecting the real identity of a subject. The use of pseudonyms, however, allows one to maintain a reference to the subject's real identity, e.g., for accountability purposes [1]. A trusted third party, adhering to agreed rules, could for instance reveal the real identities of misbehaving pseudonymous users. Pseudonymity also allows a user to link certain actions under one pseudonym. For instance, a user could re-use the same pseudonym for purchasing items at a certain online shop for building up a reputation or for collecting bonus points under this so-called relationship pseudonym.

The degree of anonymity protection provided by pseudonyms depends on the amount of personal data of the pseudonym holder that can be linked to the pseudonym, and on how often the pseudonym is used in various contexts/for various transactions. The best protection can be achieved if for each transaction a new so-called transaction pseudonym is used that is unlinkable to any other transaction pseudonyms and at least initially unlinkable to any other personal data items of its holder (see also [2]).

Cross-references

► Anonymity
► Privacy
► Privacy-enhancing Technologies

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.

Public-Key Encryption

► Asymmetric Encryption

Publish/Subscribe

Hans-Arno Jacobsen
University of Toronto, Toronto, ON, Canada

Definition

Publish/Subscribe is an interaction pattern that characterizes the exchange of messages between publishing and subscribing clients. Subscribers express interest in receiving messages and publishers simply publish messages without specifying the recipients for a message. The publish/subscribe message exchange is decoupled and anonymous. That is, publishers neither know subscribers' identities nor whether any subscribers with matching interests exist at all. This supports a many-to-many style of communication, where data sources publish and data sinks subscribe. Different classes of publish/subscribe approaches have crystallized. Their main differences lie in the way subscribers express interest in messages, in the structure and format of messages, in the architecture of the system, and in the degrees of decoupling supported. Publish/Subscribe is widely used as middleware abstraction, applied to enterprise application integration, system and network monitoring, and selective information dissemination.

Historical Background

The definition in this entry aims to be general and characterizes publish/subscribe as an *interaction pattern* that governs the interaction between *many* publishing data sources and *many* subscribing data sinks. However, in practice, publish/subscribe is often interpreted to mean many slightly different concepts, such as an asynchronous communication style, a messaging paradigm, a message routing approach, an event filtering (matching) approach, or a design pattern. Moreover, research on publish/subscribe has been conducted in different communities, such as distributed systems, networking, programming languages, software engineering, and databases. The exact origins of publish/subscribe are therefore difficult to pinpoint exactly. Today, publish/subscribe-style abstractions can be found in many messaging standards, messaging products, databases, and even in special purpose hardware solutions.

The interpretation of publish/subscribe as asynchronous communication style emphasizes the data dissemination aspect and the associated qualities of service. The primary concern is the distribution of data from *many* sources to *many* sinks. Sometimes only a single source is considered and publish/subscribe is viewed as a *one-to-many* data dissemination paradigm, analogous to multicast. The channel-based publish/subscribe model best represents this interpretation. The indirect ancestry of publish/subscribe in this context are early reliable broadcast protocols [3] and primitives for process group management in operating systems [4], which inspired work on group communication as abstraction for reliable many-to-many communication, surveyed in [5]. It is this context that resulted in the subject-based publish/subscribe approach (a.k.a. topic-based model), first articulated by Oki et al. [16] and popularized by TIBCO with its TIBCO/RV product.

The interpretation of publish/subscribe as mesaging paradigm emphasizes the asynchronously decoupled nature of publishing data sources and subscribing data sinks. This view of publish/subscribe results from the inclusion of publish/subscribe functionality into standard messaging system specifications and products, such as MQ Series from IBM or the Java Message Service specification [9]. While these abstractions focus more on asynchronous one-to-one communication realized through message queues, they also include publication, subscription and filtering capabilities, often realized as subscriber-side filtering.

The interpretation of publish/subscribe as event filtering and matching approach emphasizes the selective filtering capabilities of the approach. Subscriptions represent filter expressions and publications represent observations about events in the environment that need to be selectively brought to the attention of subscribing entities. The ancestry of this work goes back to the formulation of the *many-to-many pattern matching problem* in the artificial intelligence domain by Forgy [7] who proposed the Rete algorithm for solving this problem. Rete is the basis of many rule-based expert systems. It efficiently evaluates observed facts provided as input against rules compiled into a memory resident graph (network), or an equivalent program [7]. This work was succeeded by approaches for efficient trigger management in database systems [2,6] and research on active databases. In software engineering, similar concepts were applied for integrating software tools resulting in approaches such as YEAST [12]. Around the same time work on specifying events for event correlation and root cause analysis appeared in network management [14].

The interpretation of publish/subscribe as design pattern is based on the *Observer design pattern*, first articulated in the Gang of Four book [8], where the Observer pattern is also synonymously referred to as Publish/Subscribe. This reference is somewhat unfortunate, as the prescribed realization of the Observer pattern in the literature violates a key property of publish/subscribe. The Observer pattern is suggested for use in expressing one-to-many dependency between objects in a system. When the state of one object (the subject) changes state, all dependent objects are notified. This is achieve by having the subject know about its dependents by maintaining a list of them and requiring dependents to register, if they are interested in the subject's state changes. This violates the anonymous communication property of publish/subscribe, which requires that publishers and subscribers do not know each other.

There are a few other approach that have appeared independently in different contexts, such as tuple spaces in the programming languages context to model concurrency in the 1980's blackboard architectures in the context of artificial intelligence to model the interaction of agents, continuous query processing in the data management context, and stream processing, also in the data management context. All these approaches resemble publish/subscribe, but also differ in fundamental ways.

Foundations

A publish/subscribe system comprises *publishers*, *subscribers*, and *publish/subscribe message broker(s)*, also referred to as *message router(s)*.

Publishers and subscribers are roles held by applications built with the publish/subscribe abstraction. That is a client of the system could be publisher as well as subscriber at the same time. Subscribers express interest by registering subscriptions with the publish/subscribe system and publishers report on events by publishing messages to the publish/subscribe system. The system evaluates publications against registered subscriptions and determines which subscriptions match for a given publication. The complexity of matching varies among different publish/subscribe models. In the content-based publish/subscribe model, matching involves the evaluation of publication message content against expressive subscription filters. In the topic-based publish/subscribe model matching involves the evaluation of message topics against path-like subscription language expressions. In the channel-based publish/subscribe model, no explicit matching takes place; subscribers select among a set of channels and listen for messages broadcast on the channel.

Publications are transient and once match are not further stored or processed. Exceptions to this treatment are state-based publish/subscribe systems and the Subject spaces model [10,11] where publications might be maintained as partial matching state and as persistent state per se, respectively.

Matching and notification are performed by the publish/subscribe message brokers. In centralized installations, there is a single broker to which subscribers and publishers connect. In distributed installations, there are multiple brokers to which subscribers and publishers connect.

Publish/Subscribe offers the following decoupling characteristics. Decoupling in space allows clients to be physically distributed. Decoupling in time allows clients to be independently available. Decoupling in location means that clients do not know each others identity. This last characteristics is also referred to as anonymous communication.

A publish/subscribe system is defined by the subscription language model, the publication data model, and the matching semantic. All these elements closely depend on each other and define the subscription, the notification, the advertisement, the publication and specify the matching of the former.

The subscription language model defines the language for expressing subscriptions. It determines the expressiveness of the publish/subscribe model. For example, in the content-based model a subscription is a Boolean function over Boolean predicates. Predicates test conditions, such as equality, binary relations, or string operators over attribute values in publications. Subscriptions are also referred to as *filters*, as they specify which publication to filter out from a flow of publications processed. Some systems distinguish among the *subscriber* and the *consumer*. The subscriber is the entity that specifies subscriptions and the *consumer* is the entity that receives notifications when certain subscriptions match. Subscriber and consumer entities must not be the same.

The publication data model defines the structure, the format, and the content type of publication messages. Publications are the messages emitted by publishers. They represent the *event* of interest about the state of the system or world in the context of the modeled application. An *event* is an asynchronous state transition

of interest to subscribers. The publication is the message that conveys the occurrence of the event to any interested subscribers. In practice, the term publication and event are often used synonymously without distinguishing between the actual state transition and the message published about the event. The publication concept can be further refined by introducing *notifications*. A notification is the message sent from the publish/subscribe system to subscribers with matching subscriptions, whereas a publication is the message published by publishers. A notification must not be identical to a publication that triggered it. Systems may define a notification semantic that specified which values of a publication to forward to subscribers. Also, more refined notification semantics that apply transformations to publications before notifying subscribers are imaginable. However, many authors do not distinguish between the publication and notification concept defined above and use both terms synonymously. Some publish/subscribe approaches rely on the concept of an *advertisement*. An *advertisement* is similar to a type in programming languages or schemas in databases and specifies the kind of information a publisher will publish. It is used by publish/subscribe systems to optimize matching and routing of publications. In symmetry to the difference between subscriber and publisher, a similar difference could be made among publisher and producer, where one entity publishes, while the other entity merely advertises. However, this difference does not seem to have been explored in practice so far. Not all publish/subscribe approaches use advertisements.

The matching semantic defines the conditions under which a publication matches a subscription. For example, in content-based publish/subscribe, subscriptions are often conjuncts of Boolean predicates. That is a publication matches a subscription, if each predicate in the subscription evaluates to true given the values specified in the publication. This is a *crisp* matching semantic; it requires that the publication matches the subscription exactly by either evaluating to true or false. In contrast, an approximate matching semantic weakens the matching condition and tolerates that certain predicates do not match or only match to a certain degree. A model based on fuzzy set theory and possibility distribution is model realized in the Approximate Toronto Publish/Subscribe (A-ToPSS) project [13]. Similarly, probabilistic matching semantics that are defined by evaluating the probability that a publication matches a subscription are conceivable.

Publish/Subscribe. Table 1. Comparison of models

Model	Filtering	Publication	Subscription
Channel-based	No filtering	Messages	Listening to channels
Topic-based	Topics &topic hierarchy	Messages tagged with topics	Expressions with wildcards over topics
Type-based	Type checking	Objects	–
Content-based	Message content	Messages	Content-based filters

Also, similarity-based semantics that measure the similarity between a publication and subscription based on some similarity measure or metric are conceivable.

Various publish/subscribe models have crystallized over time. These models are the channel-based, topic-based, type-based, content-based, state-based, and subject spaces. Table 1 compares these models with respect to publications, subscriptions, and filtering capabilities they support. More details about each model and additional subject spaces and state-based smodels is provided in a separate definition for each term.

Rule-based systems are intended to process *facts* against *rules* through logical inference, forward chaining or backward chaining algorithms, or by evaluating rules on events. Facts represent the state of the world and rules represent knowledge. Rule-based systems generally require the maintenance of state in the rule engine, as multiple facts processed over time may contribute to the evaluation of rules.

Key Applications

Applications of publish/subscribe include Information dissemination, information filtering, alerting and notification.

Standards that implement the publish/subscribe are the CORBA Event Service [18], the CORBA Notification Service [19], AMQP [1], JMS [9], WS Topics, WS Notifications, WS Brokered Notifications, WS Eventing, OMG's Data Dissemination Service Specification [17], OGF's INFO-D [15].

Data Sets

The publish/subscribe research community has yet to produce benchmarks and collect data sets. Initial efforts are starting to emerge [20].

Cross-references

- ▶ Channel-Based Publish/Subscribe
- ▶ Content-Based Publish/Subscribe
- ▶ Event
- ▶ Event Routing
- ▶ State-Based Publish/Subscribe
- ▶ Streams
- ▶ Subject Spaces
- ▶ Topic-Based Publish/Subscribe
- ▶ Triggers
- ▶ Type-Based Publish/Subscribe

Recommended Reading

1. AMQP Consortium. Advanced Message Queuing Protocol Specification, version 0–10 edition, 2008.
2. Chakravarthy S. and Mishra D. Snoop: an expressive event specification language for active databases. Data Knowl. Eng., 14(1):1–26,1994.
3. Chang J.M. and Maxemchuk N.F. Reliable broadcast protocols. ACM Trans. Comput. Syst., 2(3):251–273, 1984.
4. Cheriton D.R. and Zwaenepoel W. Distributed process groups in the v kernel. ACM Trans. Comput. Syst., 3(2):77–107, 1985.
5. Chockler G.V., Keidar I., and Vitenberg R. Group communication specifications: a comprehensive study. ACM Comput. Surv., 33(4):427–469, 2001.
6. Cohen D. Compiling complex database transition triggers. ACM SIGMOD Rec., 18(2):225–234, 1989.
7. Forgy C.L. Rete: A fast algorithm for the many pattern/many object pattern match problem. Artifi. Intell., 19 (1):17–37, 1982.
8. Gamma E., Helm R., Johnson R., and Vlissides J. Design Patterns: Elements of Reusable Object-oriented Software. Addison-Wesley Longman Publishing Co., Inc., 1995.
9. Hapner M., Burridge R., and Sharma R. Java Message Service. Sun Microsystems, version 1.0.2 edition, November 9th 1999.
10. Ka Yau Leung H. Subject space: a state-persistent model for publish/subscribe systems. In Proc. Conf. of the Centre for Advanced Studies on Collaborative Research, 2002, p. 7.
11. Ka Yau Leung H. and Jacobsen H.A. Efficient matching for state-persistent publish/subscribe systems. In Proc. Conf. of the Centre for Advanced Studies on Collaborative Research, 2003, pp. 182–196.
12. Krishnamurthy B. and Rosenblum D.S. Yeast: A general purpose event-action system. IEEE Trans. Softw. Eng., 21(10):845–857, 1995.
13. Liu H. and Jacobsen H.A. Modeling uncertainties in publish/subscribe system. In Proc. 20th Int. Conf. on Data Engineering, 2004.
14. Mansouri-samani M. and Sloman M. Gem: A generalized event monitoring language for distributed systems. Distrib. Syst. Eng. J., 4:96–108,1997.
15. OGF. Information Dissemination in the Grid Environment Base Specifications, 2007.
16. Oki B., Pfluegl M., Siegel A., and Skeen D. The information bus: an architecture for extensible distributed systems. In Proc. 14th ACM Symp. on Operating System Principles, 1993, pp. 58–68.
17. OMG. Data Distribution Service for Real-time Systems, version 1.2, formal/07-01-01 edition, January 2007.
18. OMG. Event Service Specification, version 1.2, formal/04-10-02 edition, October 2004.
19. OMG. Notification Service Specification, version 1.1, formal/04-10-11 edition, October 2004.
20. The PADRES Team. Publish/subscribe data sets. http://research.msrg.utoronto.ca/Padres/DataSets, 2008.

Publish/Subscribe over Streams

Yanlei Diao[1], Michael J. Franklin[2]
[1]University of Massachusetts Amherst, MA, USA
[2]University of California-Berkeley, Berkeley, CA, USA

Definition

Publish/subscribe (pub/sub) is a many-to-many communication model that directs the flow of messages from senders to receivers based on receivers' data interests. In this model, publishers (i.e., senders) generate messages without knowing their receivers; subscribers (who are potential receivers) express their data interests, and are subsequently notified of the messages from a variety of publishers that match their interests.

Historical Background

Distributed information systems usually adopt a three-layer architecture: a presentation layer at the top, a resource management layer at the bottom, and a *middleware layer* in between that integrates disparate information systems. Traditional middleware infrastructures are tightly coupled. Publish/Subscribe [13] was proposed to overcome many problems of tight coupling:

- With respect to communication, tightly coupled systems use static point-to-point connections (e.g., remote procedure call) between senders and receivers. In particular, a sender needs to know all its receivers before sending a piece of data. Such communication does not scale to large, dynamic systems where senders and receivers join and leave frequently. Pub/sub offers loose coupling of senders and receivers by allowing them to exchange data without knowing the operational status or even the existence of each other.

- With respect to content, tight coupling can occur in remote database access. To access a database, an application needs to have precise knowledge of the database *schema* (i.e., its structure and internal data types) and is at risk of breaking when the remote database schema changes. Extensible Markup Language (XML)-based pub/sub has emerged as a solution for loose coupling at the content level. Since XML is flexible, extensible, and self-describing, it is suitable for encoding data in a generic format that senders and receivers agree upon, hence allowing them to exchange data without knowing the data representation in individual systems.

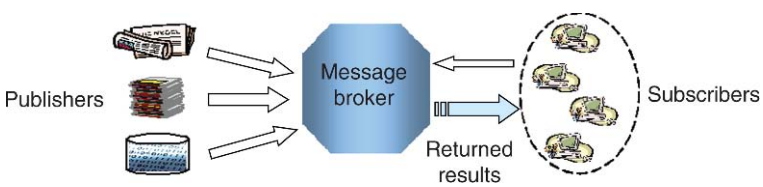
In many pub/sub systems, message brokers serve as central exchange points for data sent between systems. Figure 1 illustrates a basic context in which a broker operates. Publishers provide information by creating streams of messages (Besides "messages," the words "events," "tuples," and "documents" are often used with similar meanings in various contexts in the database literature.) that each contain a header describing application-specific information and a payload capturing the content of the message. Subscribers register their data interests with a message broker in a subscription language that the broker supports. Inside the broker, arriving subscriptions are stored as *continuous queries* that will be applied to all incoming messages. These queries remain effective until they are explicitly deleted. Incoming messages are processed on-the-fly against all stored queries. For each message, the broker determines the set of queries matched by the message. A query result is created for each matched query and delivered to its subscriber in a timely fashion.

Figure 2 shows a design space for publish/subscribe over data streams. In this diagram, pub/sub systems are first classified by the data model and the query language that these systems support. Roughly speaking, there are three main categories.

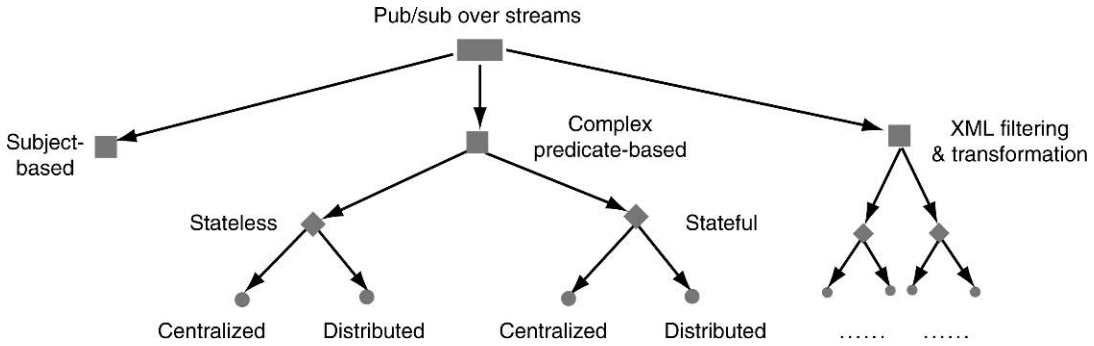
- *Subject-based*: Publishers label each message with a subject from a pre-defined set (e.g., "stock quote") or hierarchy (e.g., "sports/golf"). Users subscribe to the messages in a particular subject. These queries can also contain a filter on the data fields of the message header to refine the set of relevant messages within a particular subject.
- *Complex predicate-based*: Some pub/sub systems model the message content (payload) as a set of attribute-value pairs, and allow user queries to contain predicates connected using "and" and "or" operators to specify constraints over values of the attributes. For example, a predicate-based query applied to the stock quotes can be "Symbol = 'ABC' and (Change $>$ 1 or Volume $>$ 50,000)."
- *XML filtering and transformation*: Recent pub/sub systems have started to exploit the richness of XML-encoded messages, in particular, the hierarchical, flexible XML structure. User queries can be written using an existing XML query language such as XQuery. The rich XML structure and use of an XML query language enable potentially more accurate filtering of messages and further re-structuring of messages for customized result delivery.

Pub/sub systems can be further classified based on the style of query processing. In some systems, queries are applied only to individual messages, e.g., filtering messages, which does not involve any interaction across message boundaries. Such processing is referred to as *stateless*. Stateless processing is in contrast to stream query processing that maintains *state* over a long stream of messages, hence referred to as *stateful* processing. This distinction is illustrated for complex predicate-based systems in Fig. 2.

Finally, pub/sub systems can be distinguished based on the distribution of the architecture, as also shown in Fig. 2. In a coarse-grained fashion, this design space considers centralized and distributed processing. Distributed processing spreads the processing load for



Publish/Subscribe over Streams. Figure 1. Overview of publish/subscribe.



Publish/Subscribe over Streams. Figure 2. Design space for publish/subscribe over streams.

larger-scale pub/sub services; accordingly, it requires a more sophisticated routing functionality.

Foundations

As with stream processing, subscriptions, stored as continuous query inside a broker, need to be evaluated as data continuously arrives from other sources; that is, queries are evaluated every time when a new data item is received. Besides stream processing, pub/sub raises several additional challenges:

- *Scalability.* A key distinguishing requirement of pub/sub is scalability, in particular, in query population that pub/sub systems need to support. Such query populations can range from hundreds to millions in applications such as personalized content delivery. Given such populations, a salient issue is to efficiently search the huge set of queries to find those that can be matched by a message and to construct complete query results for them.
- *Robustness.* A second requirement of message brokers is the ability to perform in highly-dynamic environments where subscribers join and leave and their data interests change over time. Since message brokers see a constantly changing collection of queries, they must react quickly to query changes without adversely affecting the processing of incoming messages.
- *Distribution.* Due to the scale of message volume and query population, large-scale pub/sub may require the use of a network of message brokers to distribute the query population and message processing load. In this case, an additional issue is how to efficiently route a message from its publishing site to the set of brokers hosting relevant queries for complete query processing.

Scope of this entry. The rest of the entry focuses on complex predicate-based pub/sub systems. Pub/sub systems exploring XML filtering and transformation are described in detail in the entry "XML Publish/Subscribe."

Centralized, Stateless Publish/Subscribe

Le Subscribe [9] and *Xlyeme* [12] are predicate-based message filtering systems that use centralized processing. In these systems, a predicate is a comparison between an attribute and a constant using relational operators such as "=", ">", and "<". The main issue they address is how to efficiently match an incoming event, in the form of attribute value pairs, with the predicates of a large number of queries. The key idea is to *index* predicates as well as to *cluster* queries. In particular, *Le Subscribe* uses multi-attribute hash indexes to evaluate several predicates in a query with a single operation. In addition, it groups queries based on the number of contained predicates and the common conjunction of equality predicates, so many queries can be (partly) evaluated using a single operation. It further offers cost-based algorithms to find optimal clustering and to dynamically adjust it.

Centralized, Stateful Publish/Subscribe

NiagaraCQ [6] considers continuous queries with more complex predicates that can compare attributes of an input message to constants or to attributes of another message. To efficiently handle multiple queries, it groups query plans of continuous queries based on common *expression signatures*: an expression signature presents the same syntax structure, but possibly different constant values, in different queries. Consider queries that are interested in stock quotes of different symbols. Traditional query processing involves repeated

retrieval of the symbol attribute from input and evaluation of different predicates on this attribute for different queries. The group plan employs a constant table to store the constant values form different queries, and retrieves this attribute from the input once and then performs an equality join of the retrieved value and the constant table to find all matching queries. For robust processing, NiagaraCQ constructs group plans incrementally: Given a new query, it constructs the query plan, and merges the query plan bottom up with a group plan with the same signature, extending the group plan with the mismatched branch(es) if necessary. This process is incremental as the addition of the new query plan does not affect existing queries.

Recently, there has been a significant amount of activity on handling continuous and time-varying tuple streams, resulting in the development of multiple general-purpose stream management systems [1,5,11]. These systems support complex continuous queries that join multiple streams and/or compute aggregate values over a period of time called a *window* (hence, performing stateful processing). While this surge of research explores a broad set of issues such as adaptivity and approximation, shared processing of window-based queries [5,11,10] is of particular relevance to pub/sub.

Several special-purpose pub/sub systems have been recently proposed to handle temporal correlations among events in a stream. SASE [15] supports sequencing operators that integrate parameterized predicates (i.e., predicates that compare different events), negation, and windowing. It explores a new query processing abstraction that uses an automaton-based implementation for fast sequence operations and relational-style post-processing for other tasks such as negation and windowing. It also devises a set of optimizations in this automaton-based framework for efficiency and scalability. Cayuga [7] offers an algebra for expressing event sequences that may address a finite yet unbounded number of events with a similar property, and employs a more sophisticated automaton model to support this algebra. Its implementation focuses on multi-query optimization including merging states of automata for different queries and further indexing query predicates.

Distributed, Stateless Publish/Subscribe

In distributed pub/sub systems, messages are published and subscriptions are registered to different brokers. A key issue is to efficiently route each message from its publishing site to the subset of brokers hosting relevant queries for complete query processing. For complexity reasons, most distributed pub/sub systems restrict themselves to stateless services.

ONYX [8] presents an overview of a pub/sub network exploring *content-based routing*. In this paradigm, brokers are organized as an application-level overlay network with a particular topology. When a new message enters the broker network, the root broker as well as each intermediate broker routes the message to its neighboring brokers based on the correspondence between the content of the message and the subscriptions residing at and reachable from those brokers. Content-based routing is used as a key mechanism to avoid the flooding of messages to all brokers in the network, hence reducing bandwidth usage and broker processing load and rendering better scalability.

ONYX consists of two layers of functionality. The lower layer deals with the overlay network; in particular, for each broker, it constructs a broadcast tree that is rooted at that broker and reaches all other brokers in the network. On top of these broadcast trees, the higher layer performs content-based processing by dealing with subscriptions and messages. Two issues determine the effectiveness of content-based routing. The first is how to partition subscriptions and assign them to host brokers. Results of ONYX show that content-based routing is most effective if the clustering of subscriptions results in mutual exclusiveness in data interest among host brokers. The second issue is how to aggregate subscriptions from their host brokers and place such aggregations as routing specifications in the intermediate brokers for later directing the message flow. Different degrees of generalization are possible depending on the precision-efficiency tradeoff suitable for each pub/sub system.

For content-based routing, Gryphon [2] and Siena [3] both aggregate subscriptions into compact, precise in-network data structures and use efficient algorithms to search these data structures to determine the routing of the messages to other brokers in the network.

XPORT [14] focuses on the construction, maintenance, and optimization of an overlay dissemination tree of the available message brokers in the system. Its tree-oriented optimization framework consists of a generic aggregation model that allows system cost to be expressed through various combinations of aggregation functions and local metrics, and distributed

iterative optimization protocols for cost-based optimization of the overlay structure.

Distributed, Stateful Publish/Subscribe

For stateful publish/subscribe, Chandramouli et al. [4] adopt the following model: Users define subscriptions as SQL views over a conceptual (possibly distributed) database and messages are published as updates to the database; if a database update affects a subscription, the pub/sub system sends a notification to the subscriber containing the change to the content of the subscription view. The main idea of this work is to explore appropriate interfacing between the database and the pub/sub network so that existing stateless pub/sub networks can be leveraged for efficient dissemination. To do so, the key is to transform published messages into a semantic description of affected subscriptions (performed at the database side) and subscriptions into a predicate over the semantic description (evaluated in the stateless pub/sub network). Consider a selection-join subscription $\sigma_p(\sigma_{p\text{-}R}\ R \rhd\lhd \sigma_{p\text{-}S}\ S)$. If a new message applies an update ΔR to table R, its effect on the subscription, $\sigma_p(\sigma_{p\text{-}R}\ \Delta R \rhd\lhd \sigma_{p\text{-}S}\ S)$, requires access to table S that is not in the original update message (hence, stateful processing). The proposed solution reformulates each message ΔR into a series of messages containing the tuples in $\Delta R \rhd\lhd S$ at the database side; to utilize a stateless pub/sub network, it transforms the select-join subscription into a simple condition that evaluates $\sigma_{p \wedge p\text{-}R \wedge p\text{-}S}$ over reformulated messages.

Key Applications

Personalized content delivery. This class of applications provide personalized filtering and delivery of news feeds, web feeds (RSS), stock tickers, sport tickers, etc. to large numbers of online users.

Online auction and online procurement. Through these applications, users can create their own feeds for their favorite searches, for example, on eBay.

Enterprise information management. Pub/sub has been traditionally used to support application integration, which integrates disparate, independently-developed applications into new services via the loose coupling of senders and receivers based on receivers' data interest.

System and network monitoring. Pub/sub has been recently used in system and network monitoring, where large-scale complex systems generate reports categorizing various aspects of system performance and resource utilization, and system administrators, end users, and visualization applications subscribe to receive updates on particular aspects of those reports.

Data Sets

Many data sources are available online, including RSS feeds (indicated by the orange button labeled with "RSS" or "XML" in many web pages) and financial feeds (e.g., Yahoo! Finance).

Cross-references

- ▶ Continuous Query
- ▶ Stream Processing
- ▶ XML
- ▶ XML Document
- ▶ XML Publish/Subscribe
- ▶ XPath/XQuery

Recommended Reading

1. Abadi D., Carney D., Cetintemel U., Cherniack M., Convey C., Lee S., Stonebraker M., Tatbul N., and Zdonik S. Aurora: a new model and architecture for data stream management. VLDB J., 12(2):120–139, 2003.
2. Aguilera M.K., Strom R.E., Sturman D.C., Astley M., and Chandra T.D. Matching events in a content-based subscription system. In Proc. ACM SIGACT-SIGOPS 18th Symp. on the Principles of Dist. Comp., 1999.
3. Carzaniga A. and Wolf A.L. Forwarding in a content-based network. In Proc. ACM Int. Conf. of the on Data Communication, 2003, pp. 163–174.
4. Chandramouli B., Xie J., and Yang J. On the database/network interface in large-scale publish/subscribe systems. In Proc. ACM SIGMOD Int. Conf on Management of Data, 2006, pp. 587–598.
5. 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.
6. Chen J., Dewitt D.J., Tian F., and Wang Y. NiagaraCQ: a scalable continuous query system for Internet databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 379–390.
7. Demers A.J., Gehrke J., Hong M., Riedewald M., and White W.M. Towards expressive publish/subscribe systems. In Advances in Database Technology, Proc. 10th Int. Conf. on Extending Database Technology, 2006, pp. 627–644.
8. Diao Y., Rizvi S., and Franklin M.J. Towards an Internet-scale XML dissemination service. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 612–623.
9. Fabret F., Jacobsen H.A., Llirbat F., Pereira J., Ross K.A., and Shasha D. Filtering algorithms and implementation for very fast

publish/subscribe systems. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 115–126.
10. Krishnamurthy S. Shared query processing in data streaming systems. Ph.D. Dissertation, University of California, Berkeley.
11. 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.
12. Nguyen B., Abiteboul S., Cobena G., and Preda M. Monitoring XML data on the Web. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 437–448.
13. Oki B., Pfleugl M., Siegel A., and Skeen D. The information bus: an architecture for extensible distributed system. In Proc. 14th ACM Symp. on Operating System Principles, 1993, pp. 58–68.
14. Papaemmanouil O., Ahmad Y., Çetintemel U., Jannotti J., and Yildirim Y. Extensible optimization in overlay dissemination trees. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 611–622.
15. Wu E., Diao Y., and Rizvi S. High-performance complex event processing over streams. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 407–418.

Punctuations

David Maier[1], Peter A. Tucker[2]
[1]Portland State University, Portland, OR, USA
[2]Whitworth University, Spokane, WA, USA

Definition

A *punctuation* is an item embedded in a data stream that specifies a subset of the domain of that stream. A data item that belongs to the subset specified by a punctuation is said to *match* that punctuation. A data stream is said to be *grammatical* if, for each punctuation, no data items will follow that match that punctuation. Consider a stream of bids for online auctions. When an auction closes, a punctuation p is embedded in the stream that matches all bids for that auction. In a grammatical stream, p indicates no more bids will arrive from that stream for that auction.

In the most common format, a punctuation is a tuple of *patterns*, where each pattern corresponds to an attribute of the data items in a stream. Typically, four patterns are used: a *wildcard* (denoted by '∗') is matched by all values, a *literal* is matched by only the given value, a *list* (denoted by { }) is matched by any value in the given list, and a *range* (denoted by [ ]) is matched by any value that falls in the given range. If each value in a data item matches its corresponding pattern in a punctuation, then the data item matches the punctuation. For example, if auction bids have schema <*auction_id*, *bidder_id*,*price*,*timestamp*>, given the punctuation p = P<{ 105,106},∗,[0.00,100.00],∗>, all bids for auctions 105 and 106 with prices between 0 and 100 match p. Thus, the data item <105,95,90.00,10495> matches p, but the data items <104,95,90.00,10495> and <105,95,105.00,10495> do not.

Key Points

Punctuations have been shown to unblock query operators and reduce the amount of state required by query operators [2] when processing non-terminating data streams. In addition, punctuations have proven useful for dealing with disorder in streams [1] and in specifying window semantics [1,3].

A *punctuation-aware operator* is a stream query operator that can take advantage of grammatical streams. Such an operator implements three different behaviors in the presence of punctuations [2]. A *pass behavior* allows operators to output data items due to punctuations. For example, when a punctuation arrives, an aggregate can output results for a particular group if all possible data items for that group match that punctuation. A *keep behavior* specifies which data items must remain in state when a punctuation arrives. For example, when a punctuation arrives, some data items that are held in state for symmetric hash join may be released. Finally, a *propagation behavior* defines which punctuations can be output when punctuation arrives. For example, union can output any punctuation that has arrived from both inputs.

Most punctuation-aware operators are counterparts of operators on finite relations. However, there can be more than one way to define pass, keep and propagation behavior when extending a relational operator to be punctuation-aware. An important property here is for a stream operator s to be *faithful* to its analogous relational operator r, which means that on any finite prefix x of a stream, the output of s on x is consistent with the output of r on any finite extension of x.

Cross-references

▶ Continuous Query
▶ Data Stream
▶ Stream-Oriented Query Languages and Operators
▶ Stream Models
▶ Stream Processing
▶ Window-Based Query Processing

Recommended Reading

1. Li J., Maier D., Tufte K., Papadimos V., and Tucker P.A. Semantics and evaluation techniques for window aggregates in data streams. In ACM SIGMOD International Conference on Management of Data, 2005, pp. 311–322.
2. Tucker P.A., Maier D., Sheard T., and Fegaras L. Exploiting punctuation semantics in continuous data streams. IEEE Trans. Knowl. Data Eng., 15(3):555–568, 2003.
3. Tucker P.A., Maier D., Sheard T., and Stephens P. Using punctuation schemes to characterize strategies for querying over data streams. IEEE Trans. Knowl. Data Eng., 19(9):1227–1240, 2007.

Push Transactions

► Information Filtering

Push/Pull Delivery

► Data Broadcasting, Caching and Replication