R

RAID

► Redundant Array of Independent Disks

Random Access Memory (RAM)

► Main Memory

Randomization Methods to Ensure Data Privacy

Ashwin Machanavajjhala, Johannes Gehrke
Cornell University, Ithaca, NY, USA

Synonyms

Perturbation techniques

Definition

Many organizations, e.g., government statistical offices and search engine companies, collect potentially sensitive information regarding individuals either to publish this data for research, or in return for useful services. While some data collection organizations, like the census, are legally required not to breach the privacy of the individuals, other data collection organizations may not be trusted to uphold privacy. Hence, if U denotes the original data containing sensitive information about a set of individuals, then an untrusted data collector or researcher should only have access to an *anonymized* version of the data, $U^\star$, that does not disclose the sensitive information about the individuals. A randomized anonymization algorithm R is said to be a *privacy preserving randomization method* if for every table T, and for every output $T^\star = R(T)$, the privacy of all the sensitive information of each individual in the original data is provably guaranteed.

Historical Background

This is a brief survey of state of the art randomization methods. The reader is referred to the classical survey by Adam and Wortman [1] for a more comprehensive description of older randomization techniques.

Existing literature can be classified based on whether the individuals sharing the data trust the data collector or not. The untrusted data collector scenario is discussed first. Randomization methods have been historically used to elicit accurate answers to surveys of sensitive `yes/no` questions. Respondents may be reluctant to answer such questions truthfully when the data collector (surveyor) is untrusted. Warner's classical paper on *randomized response* [17] proposed a simple technique, where each individual i independently randomized the answer as follows: i answers truthfully with probability p_i, and lies with probability $(1 - p_i)$. Randomized response intuitively ensures privacy since no individual reports the true value. However, Warner did not formalize this intuition.

Subsequent works [7,2] generalized the above randomized response technique to other domains. Evfimievski et al. [7] studied the problem where individuals share itemsets (e.g., a set of movies rented) with an untrusted server (e.g., an online movie rental company) in return for services (e.g., movie recommendations), and they proposed a formal definition of privacy breaches. They invented a provably private randomization technique where users submit independently randomized itemsets to the server. They also proposed data reconstruction algorithms to help the server mine association rules from these randomized itemsets, and experimentally illustrated the accuracy of the reconstruction techniques. The above methods are called *local randomization techniques* since every individual perturbs his/her data locally before sharing it with the data collector.

In the trusted data collector scenarios, while the individuals trust the data collector, they may not trust any third party with whom their data is shared. Hence, randomization methods have been proposed to help

privately share the collected data. These techniques can be broadly categorized into *input randomization* and *output randomization* techniques. Input randomization techniques publish a perturbed version of the table; queries are answered using the perturbed data. Output randomization techniques, on the other hand, execute queries on the real data, and return perturbed answers.

One thread of work on input randomization techniques was initiated by Agrawal and Srikant [3]. They proposed an input randomization technique wherein 0-mean random noise is added to the numeric attributes of each individual in the table. The algorithm was experimentally shown to be utility preserving. Nevertheless, Kargupta et al. [10] and Huang et al. [9] showed that adding noise independently to each record in the table does not guarantee privacy.

Yet another thread of work involves publishing synthetic data that has the same properties as the original data, but preserves privacy. Synthetic data generation is a very popular technique in the statistics community, and real applications (like OnTheMap [13]) publish sensitive information using this technique. First proposed by Rubin [16], these techniques build a statistical model using a noise infused version of the data, and then generate synthetic data by randomly sampling from this model. While much research has focused on deriving variance and confidence estimators from synthetic data, only recently has the privacy of these techniques been formally analyzed [12,15].

Among output randomization techniques, the SULQ framework proposed by Blum et al. [5] stands out since it has provable guarantees of privacy. Here, numeric query answers are perturbed by adding Laplace noise. Unlike in input perturbation techniques, privacy is guaranteed if and only if the number of queries that are answered is sub-linear in the number of entities in the table. Nevertheless, Blum et al. show that a large number of useful data mining tasks can be performed using this framework. However, exploratory research could be hindered in this framework, since the researchers need to formulate their queries before seeing the data.

Foundations

Local Randomization Techniques

Let U be the original data and let D_U be its (potentially multi-dimensional) domain. Each record $u \in U$ corresponds to the sensitive information of a distinct individual. Each u is independently randomized using a perturbation matrix $\mathbf{A}$. The entry $\mathbf{A}[u, v]$ describes the transition probability $\Pr[u \rightarrow v]$ of perturbing a record $u \in D_U$ to a value v in the perturbed domain D_V. This random process maps to a Markov process, and the perturbation matrix $\mathbf{A}$ should therefore satisfy the following properties:

$$\mathrm{A} \geq 0, \qquad \sum_{v \in D_V} \mathbf{A}[u, v] = 1 \ \forall_u \in D_U \tag{1}$$

Privacy

Since each record $u \in U$ is perturbed independent of the rest of the records, it is sufficient to reason about the privacy of each record separately. A *privacy breach* [7] is said to occur if for some predicate ϕ of an individual's private information, the prior belief in the truth of ϕ is very different from the posterior belief in its truth after seeing the randomized record $R(u)$. More precisely, an *upward* (ρ_1, ρ_2) *privacy breach* with respect to a predicate ϕ occurs if

$$\begin{aligned} &\exists u \in U, \exists v \in Dv, \text{ s.t.,} \\ &\Pr[\phi(u)] \leq \rho_1 \text{ and } Pr[\phi(u)|R(u) = v] \geq \rho_2 \end{aligned} \tag{2}$$

Similarly, a *downward* (ρ_1, ρ_2) *privacy breach* with respect to a predicate ϕ occurs if

$$\begin{aligned} &\exists u \in U, \exists v \in Dv, \text{ s.t.,} \\ &\Pr[\phi(u)] \geq \rho_1 \text{ and } \Pr[\phi(u)|R(u) = v] \leq \rho_2 \end{aligned} \tag{3}$$

A randomization method R is defined to be γ-*amplifying* if

$$\forall v \in D_V, \ \forall u_1, u_2 \in D_U, \ \frac{Pr[u_1 \rightarrow v]}{Pr[u_2 \rightarrow v]} \leq \gamma \tag{4}$$

Evfimievski et al. [8] showed that a local randomization method that is γ-amplifying permits a (ρ_1, ρ_2) privacy breach if and only if

$$\frac{\rho_2}{\rho_1} \cdot \frac{1 - \rho_1}{1 - \rho_2} \geq \gamma \tag{5}$$

Algorithms

Randomized Response: Warner's randomized response technique [17] can be instantiated in this model as follows. Each entry $u \in U$ is a *yes/no* answer given by a distinct individual to a sensitive question Q (e.g.,

"Have you ever used illegal drugs?"). Hence, $D_U = \{0,1\}$. In order to preserve privacy, each individual flips a coin with bias p, and answers honestly if the coin lands heads and lies otherwise. The perturbation matrix is the 2×2 matrix with $\mathbf{A}[0,0] = \mathbf{A}[1,1] = p$ and $\mathbf{A}[0,1] = \mathbf{A}[1,0] = (1-p)$.

Given n such perturbed answers, the aggregate answer can be estimated as follows. Let π be the fraction of the population for which the true response to Q is *yes* . Then the expected proportion of *yes* responses is

$$\Pr[\text{yes}] = \pi \cdot p + (1-\pi) \cdot (1-p) \tag{6}$$

$$\text{Hence, } \pi = \frac{\Pr[\text{yes}] - (1-p)}{2p-1} \tag{7}$$

If m out of the n individuals answered *yes* , then the following $\hat{\pi}$ is an unbiased estimator for π.

$$\hat{\pi} = \frac{\frac{m}{n} - (1-p)}{2p-1} \tag{8}$$

Warner also proposed a second randomization technique wherein, instead of lying with probability $(1-p)$, the respondent answers the question Q honestly with probability p and answers a different innocuous question Q_I with probability $(1-p)$. For instance, with probability p, the respondent truly answers if she had used illegal drugs, and with probability $(1-p)$, the respondent flips a coin with bias α and answers *yes* if the respondent got a head. In this case, the probability that the answer to Q_I is *yes* is α. Hence, if m out of the n individuals answered *yes* , then the following $\bar{\pi}$ is an estimator for π.

$$\Pr[\text{yes}] = \pi \cdot p + \alpha \cdot (1-p) \tag{9}$$

$$\pi = \frac{\Pr[\text{yes}] - (1-p) \cdot \alpha}{p} \tag{10}$$

$$\bar{\pi} = \frac{\frac{m}{n} - (1-p) \cdot \alpha}{p} \tag{11}$$

Typically, the innocuous question method is better than the former method, since the estimator $\bar{\pi}$ has a smaller variance than $\hat{\pi}$ when the probability of answering the correct question p is not too small.

Itemset Randomization

Itemset randomization is a useful tool that allows users to privately share their (e.g., shopping) histories with a centralized server in return for recommendation services. Let $\mathcal{I}$ represents the set of all items (e.g., products bought by at least one of the users). Then each entry $u \in U$ corresponds to a set of items $\mathcal{I}_u \subseteq \mathcal{I}$. Suppose, for simplicity, all the itemsets $\mathcal{I}_u$ are assumed to have the same number of items, say m. The server wants to learn the *frequent itemsets*, i.e., itemsets $A \subseteq \mathcal{I}$ whose support $\left(sup(A) := \frac{|\{u \in U | A \subseteq \mathcal{I}_u\}|}{|U|}\right)$ is $\geq s_{min}$. The following *Select-a-Size* algorithm, with parameters ρ and $\{p[j]\}_j^m = 0$, is used to randomize itemsets.

- Select an integer $j \in [1,m]$, with probability $p[j]$.
- Select a simple random sample of size j of $\mathcal{I}_u$, called $\mathcal{I}'_u$.
- For every $a \in \mathcal{I} - \mathcal{I}_u$, add a to $\mathcal{I}'_u$ with probability ρ.

Evfimievski et al. [8] proved sufficient conditions on the parameters, ρ and $\{p[j]\}_j^m = 0$, in order for this algorithms to be γ-amplifying while simultaneously maximizing the utility of the randomization method (e.g., maximizing the number of original items retained in the randomized itemset, $|I_u \cap I'_u|$). Algorithms for recovering the original data from the randomized itemsets and unbiased estimators for the mean and the covariance of these estimates are provided in [8].

Output Perturbation Techniques

Again let U be the original data. Output perturbation techniques execute queries on the real data, and then return perturbed answers. More precisely, if Q is a query on the data U, and R is the perturbation algorithm, $R(Q(U))$ is returned as the answer. In any such technique, there needs to be a limit on the number and the type of queries that can be posed to the database; for instance, answering the same query Q a large number of times discloses the exact answer to Q.

Privacy

Recall that each record in U corresponds to a distinct individual, and that the value of every record in U is independent of the other records. If each query accesses only one record in the table, then output perturbation essentially reduces to local randomization and the privacy breach analysis can be used.

When a query Q accesses multiple records ($U_Q \subseteq U$), however, one cannot reason about the privacy of a single record $u \in U$ (and hence, the privacy of an individual i) in isolation from the rest of the records $U_Q - \{u\}$. Moreover, depending on the amount of prior

information known about U_Q, the extent of u's disclosure varies. For instance, if u is, say, the salary of Betty, Q is the query returning the total salary of women in the table, and the adversary knows that Betty is the only woman in the department ($|U_Q| = 1$), then disclosing $Q(U)$ discloses the value of u completely.

Differential privacy [6] can be used to quantify privacy in this case. Answering a query $Q(.)$ using R ($Q(.)$) is ε- *differential private* if for every pair of original tables U_1 and U_2 that differ in the value of a single record u, and for every possible answer A,

$$\left|\log\left(\frac{\Pr[R(Q(U_1)) = A]}{\Pr[R(Q(U_2)) = A]}\right)\right| \leq \epsilon \qquad (12)$$

Intuitively, the above definition preserves privacy as follows. Consider a worst-case adversary who knows the exact values of all the records in $U - \{u\}$ and who is attempting to discover the value of record u_i. Now, if α_S denote the adversary's prior belief that $u \in S$ ($S \subseteq D_U$), then after seeing the answer $R(Q(U))$, the adversary's posterior belief β_S conditional on his knowledge of the rest of the records in the table is bounded by,

$$\alpha_S / e^{\epsilon} \leq \beta_S \leq e^{\epsilon} \times \alpha_S \qquad (13)$$

Algorithms

The SULQ framework, introduced by Blum et al. [6], answers aggregate queries by adding random noise. Let Q be a function $D_U^n \rightarrow \Re$ The *sensitivity* of query Q, is the smallest number $S(Q)$, such that

$$\begin{aligned} &\forall U_1, U_2 \text{that differ in one record,} \\ &|Q(U_1) - Q(U_2)| \leq S(Q) \end{aligned} \qquad (14)$$

Let Lap(λ) denote the *Laplace* distribution which has a density function $h(y) \propto \exp(-|y|/\lambda)$. Suppose a query $Q(U)$ posed to a database U is answered using $Q(U) + Y$, where $Y \sim$Lap($S(Q)/\varepsilon$). This perturbation scheme satisfies ε-differential privacy. For every U_1, U_2 that differ in only one record u,

$$\begin{aligned} \frac{\Pr[Q(U_1) + Y = x]}{\Pr[Q(U_2) + Y = x]} &= \frac{h(x - Q(U_1))}{h(x - Q(U_2))} \\ &= \frac{\exp(-|x - Q(U_1)| \times \epsilon / S(Q))}{\exp(-|x - Q(U_2)| \times \epsilon / S(Q))} \end{aligned} \qquad (15)$$

$$\leq \exp(\epsilon \times |Q(U_1) - Q(U_2)|/S(Q)) = \exp(\epsilon) \qquad (16)$$

This technique is useful when the amount of noise added is small; i.e., when the Q has low sensitivity. Examples of queries with low sensitivity are histograms, linear aggregation queries.

Input Perturbation Techniques

Most research is exploratory; the output perturbation techniques can be inconvenient, since researchers must specify queries before seeing the data. Input perturbation techniques on the other hand publish a perturbed version of the data that the researchers can then directly query. Though the two techniques seem different, technically they are the same; input perturbation uses a single perturbed query over the data, namely the sanitization algorithm. Dwork et al. [6] and Kifer et al. [11] show that in many cases publishing perturbed answers to multiple queries gives more utility than publishing a single perturbed dataset like in input perturbation.

Privacy

A simple technique to perturb the data is to independently add 0-mean noise to each attribute of each record. Let V be a noise matrix, then the perturbed data is $U_p = U + V$. The random noise added to each cell ($v \in V$) is usually either a uniform random variable in $[-\alpha,\alpha]$ or distributed as a Gaussian with 0 mean and a known variance. The privacy of such a scheme is unclear; in fact, Kargupta et al. [10] and Huang et al. [9] showed that the very accurate estimates of the original data can be recovered from such additively perturbed data due to dependencies inherent in U. For instance, suppose an adversary knows that all the records in U have the same value, say z. Then, additive randomization does not guarantee any privacy; the mean of the perturbed data accurately estimates z if there are enough records in U.

Additive randomization can be broken using Principal Components Analysis (PCA). Suppose the data has m dimensions and is perturbed by adding noise independently to each dimension. Usually, different attributes in the data are correlated; hence, it can projected onto a smaller number, $p < m$, of dimensions. The first principle component (PC) of the data is the direction, e_1, along which the data has the highest variance. The i^{th} PC, e_i, is a vector orthogonal to the first $(i - 1)$ PC's with the largest variance. These vectors are the eigenvectors of the covariance matrix of the data. In correlated data, only the variances along p directions are large. However, for the random data, the variances are the same along all directions. The variances of the perturbed data are roughly the sum

of the variances of the original data and the random noise. Hence, by dropping $(m - p)$ directions along which the perturbed data has the least variance, while much information is not lost about the original data, a $(1 - p/m)$ fraction of the noise added is removed; this might lead to privacy breaches.

Algorithms

In order to guarantee privacy, the noise added should be correlated to the data. This is ensured by *synthetic data generation*. Here, a statistical model is generated from a noise infused version of the existing data, and synthetic data points are sampled from this model. Noise is introduced into the synthetic data on two counts: the noise infused prior to building the model, and the noise due to random sampling.

Different algorithms for generating synthetic data can be created by varying the synthetic model that is built using the data. One simple technique is based on Dirichlet resampling [12]. Let H denote the histogram of U, i.e., $H = \{f(v) |\ v \in D_U, f(v) =$ multiplicity of v in U$\}$, and let R denote the noise histogram. Then the statistical model is $D(H + R)$, where D denotes the Dirichlet distribution. Synthetic data is generated as follows. Draw a vector of probabilities, X, from $D(H + R)$, and generate m points according to the probabilities in X. The above process is mathematically equivalent to the following resampling technique. Consider an urn with balls marked with values $v \in D_U$ such that the number of balls marked with v equals the sum of the frequency of v in U and the frequency of v in the noise histogram. Synthetic data is generated in m sampling steps as follows. In each sampling step, a ball, say marked v, is drawn at random and two balls marked v are added back to the urn. In this step, the synthetic data point is v.

Machanavajjhala et al. [12] characterized the privacy guaranteed by this algorithm in terms of noise distribution. Specifically, they showed that in order to guarantee ε-differential privacy, the frequency of every $v \in D_U$ in the noise histogram should be at least $m/(e^{\varepsilon} - 1)$. For large m and small ε the noise required for privacy overwhelms all of the signal in the data and renders the synthetic data completely useless. Such large requirements of noise is due to the following worst case requirement of differential privacy. Consider a scenario where an adversary knows that U contains exactly one record u_i that takes either the value v_1 or v_2. Now suppose that in the output sample, every record takes the value v_1. If m is large, then the adversary's belief that $r_u = v_1$ is close to 1. In order to guard against such adversaries, differential privacy requires a large amount of noise. However, the probability that such synthetic data is output is negligibly small. This can be remedied using a weaker (ε, δ)-probabilistic differential privacy definition, where an algorithm is private if it satisfies ε-differential privacy for all outputs that are generated with a cumulative probability of at least $(1 - \delta)$. Under this weaker definition, the Dirichlet resampling technique is private with much smaller noise requirements.

Barak et al. [4] propose a solution to publish marginals of a contingency table (i.e., a histogram) using the SULQ framework. Publishing a set of noise infused marginals is not satisfactory; such marginals may not be consistent, i.e., there may not exist a contingency table that satisfies all these marginal contingency tables. Barak et al. solve this problem by adding noise to a small number of Fourier coefficients; any set of Fourier coefficients correspond to a (fractional and possibly negative) contingency table. They show that only a "small" number of Fourier coefficients are required to generate the required marginals, and hence only a small amount of noise (proportional to the size of the marginal domain) is required. The authors employ a linear program solution (in time polynomial in the size of multidimensional domain) to generate the final non-negative integral set of noise infused marginals.

Rastogi et al. [14] propose the $\alpha\beta$ algorithm for publishing itemsets. It is similar to the select-a-size randomization operator. Given an itemset I that is a subset of the domain of all items D, the $\alpha\beta$ algorithm creates a randomized itemset V by retaining items in I with probability $\alpha + \beta$ and adding items in $D - I$ with probability β. This algorithm satisfies a variant of ε-differential privacy. Moreover, the authors show that for queries $Q : 2^D \rightarrow \mathbb{R}$, $Q(I)$ can be estimated as follows:

$$\hat{Q}(I) = (Q(V) - \beta Q(D))/\alpha \qquad (17)$$

where $Q(V)$ and $Q(D)$ are the answers to the query Q on the randomized itemset V and the full domain D, respectively. $\hat{Q}(I)$ is shown to provably approximate $Q(I)$ with high probability.

Key Applications

Privacy preserving techniques have been used in Census applications and various web-applications.

The Dirichlet resampling based synthetic data generation technique is used in the web-based OnTheMap Census application that plots worker commute patterns on the U.S. map to study workforce indicators [13]. Warner's randomized response has been used in eliciting responses to sensitive survey questions.

Future Directions

One common property of all provably private randomization methods is that the probability of perturbing a value $u \in D_U$ to every value $v \in D_V$ (the perturbed domain, which is usually the same as D_U) should be positive. As shown by Machanavajjhala et al. [12], this causes a problem when the size of the domain is very large. For instance, in the On The Map [13] application, the domain D_U is the set of census blocks on the U.S. map and there are about 8 million such blocks. However, given a destination, there is only a few hundred workers commuting to it. Hence, even if a small amount of noise is added to each block on the map, spurious commute patterns will arise in the synthetic data. All of the techniques discussed in this article should be revisited in the context of real scenarios with sparse data.

Cross-references

- ▶ Association Rule Mining on Streams
- ▶ Differential Privacy
- ▶ Principal Component Analysis
- ▶ Statistical Disclosure Limitation for Data Access
- ▶ Synthetic Data

Recommended Reading

1. Adam N.R. and Wortmann J.C. Security-control methods for statistical databases: a comparative study. ACM Comput. Surv., 21(4):515–556, 1989.
2. Agrawal R. and Srikant R. Privacy preserving data mining. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 439–450.
3. Agrawal S. and Haritsa J.R. A framework for high-accuracy privacy-preserving mining. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 193–204.
4. Barak B., Chaudhuri K., Dwork C., Kale S., McSherry F., and Talwar K. Privacy, accuracy and consistency too: a holistic solution to contingency table release. In Proc. 26th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2007.
5. Blum A., Dwork C., McSherry F., and Nissim K. Practical privacy: the SuLQ framework. In Proc. 24th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2005, pp. 128–138.
6. Dwork C., McSherry F., Nissim K., and Smith A. Calibrating noise to sensitivity in private data analysis. In Proc. 3rd Theory of Cryptography Conf., 2006, pp. 265–284.
7. Evfimievski A., Gehrke J., and Srikant R. Limiting privacy breaches in privacy preserving data mining. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2003, pp. 211–222.
8. Evfimievsky A., Srikant R., Gehrke J., and Agrawal R. Privacy preserving data mining of association rules. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 217–228.
9. Huang Z., Du W., and Chen B. Deriving private information from randomized data. In Proc. 23th ACM SIGMOD Conf. on Management of Data, 2004.
10. Kargupta H., Datta S., Wang Q., and Sivakumar K. On the privacy preserving properties of random data perturbation techniques. In Proc. 2003 IEEE Int. Conf. on Data Mining, 2003, pp. 99–106.
11. Kifer D. and Gehrke J. Injecting utility into anonymized datasets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006.
12. Machanavajjhala A., Kifer D., Abowd J., Gehrke J., and Vihuber L. Privacy: from theory to practice on the map. In Proc. 24th Int. Conf. on Data Engineering, 2008.
13. On The Map (Version 2) http://lehdmap2.dsd.census.gov/.
14. Rastogi V., Suciu D., and Hong S. The Boundary Between Privacy and Utility in Data Publishing. Tech. rep., University of Washington, 2007.
15. Reiter J. Estimating risks of identification disclosure for microdata. J. Am. Stat. Assoc., 100:1103–1113, 2005.
16. Rubin D.B. Discussion statistical disclosure limitation. J. Off. Stat., 9(2):461–468, 1993.
17. Warner S.L. Randomized response: a survey technique for eliminating evasive answer bias. J. Am. Stat. Assoc., 60(309): 63–69, 1965.

Range Partitioning

- ▶ Physical Database Design for Relational Databases

Range Query

Mirella M. Moro
The Federal University of Rio Grande do Sul, Porto Alegre, Brazil

Synonyms

Range search; Range selection

Definition

Consider a relation R with some numeric attribute A taking values over an (ordered) domain D. A *range query*

retrieves all tuples in R whose attribute A has values in the interval $[low, high]$. That is, $low \leq R.A \leq high$. The range interval may be closed as above, open (e.g., $low < R.A < high$), or half-open in either side (e.g., $low < R.A \leq high$). A range query can be one-sided (e.g., low $\leq R.A$ retrieves all tuples with R.A value greater or equal to *low*). When $low = high$ the range query becomes an *equality* (or *membership*) query.

Key Points

Range queries involve *numeric* (or numerical) attributes. These are attributes whose domain is totally ordered and thus a query interval (e.g., *[low, high]*) can be formed. In contrast, attributes whose domain is not naturally ordered are called *categorical* (or *nominal*). Range queries correspond to selections and are thus amenable to indexing. The standard access method for a range query on some attribute A is a B+-tree built on the values of attribute A. Since the B+-tree maintains the order of the indexed values in its leaf pages, a range query is implemented as a search for the leaf page with the lower value of the range interval, followed by the accessing of sibling pages until a page that contains the higher value of the range interval is reached. The above discussion considers the one-dimensional range search. Multidimensional range queries are also important. A typical example is the *spatial range query* that retrieves all objects which fall within (or intersect, overlap, etc.) a region (a rectangle specified by ranges in each dimension). Such multidimensional range queries are typically indexed by R-trees.

Cross-references

- ▶ Access Methods
- ▶ B+-Tree
- ▶ Indexing
- ▶ Rtree
- ▶ Spatial Range Query

Range Search

- ▶ Range Query

Range Selection

- ▶ Range Query

Rank Swapping

- ▶ Data/Rank Swapping

Ranked Multimedia Retrieval

- ▶ Top-k Selection Queries on Multimedia Datasets

Ranked XML Processing

Amélie Marian[1], Ralf Schenkel[2], Martin Theobald[3]
[1]Rutgers University, Piscataway, NJ, USA
[2]Max Planck Institute for Computer Science, Saarbrücken, Germany
[3]Stanford University, Stanford, CA, USA

Synonyms

Aggregation and threshold algorithms for XML; Approximate XML querying; Top-k XML query processing

Definition

When querying collections of XML documents with heterogeneous or complex schemas, existing query languages like XPath or XQuery with their exact-match semantics are often not the perfect choice. Such exact querying languages will typically miss many relevant results that do not conform to the strict formulation of the query.

Top-k query processing for XML data, which focuses on finding the k top-ranked XML elements to an XPath (or XQuery) query with full-text search predicates, is a particularly appropriate query model for querying semi-structured data when the actual content or structure of the underlying data is not fully known. Challenges in processing top-k queries over XML data include scoring individual answers based on how closely they match the query, supporting IR-style vague search over both content and structure, and ranking the k best answers in an efficient manner.

Historical Background

Non-schematic XML data that comes from many different sources and inevitably exhibits heterogeneous structure and annotations in the form of hierarchical

tags and deeply nested XML elements often cannot be adequately searched using pure database-style query languages like XPath or XQuery. Typically, queries either return too many or too few results using only Boolean search predicates. Rather, the ranked-retrieval paradigm needs to be called for, with relaxable search conditions, various forms of similarity predicates on tags and contents, and quantitative relevance scoring. The information retrieval (IR) community has historically focused on scoring documents based on how closely they match a user's keyword query. Intense research on applying IR techniques to XML data has started in the early 2000's and has meanwhile gained considerable attention. Recent IR extensions to XML query languages such as XPath 1.0 Full-Text or the NEXI query language used in the INEX benchmark series [5] reflect this emerging interest in IR-style ranked retrieval over semi-structured data. So far, various work on scoring answers to XML queries have focused on adapting IR-style scoring techniques from unstructured text to the semi-structured world, see, e.g., [2,5,11]. On the IR side, some foray into adding structure to standard IR search has taken place before the advent of XML [13]. The popularity of XML provides an opportunity to combine efforts led separately by both the DB and IR communities and provide robust techniques to query semi-structured data.

Threshold Algorithms

The method of choice for efficient processing of top-k similarity queries is the family of threshold algorithms (TA), most notably presented by Fagin et al. [8] and originally developed for multimedia databases and structured records stored in relational database systems (RDBMS). These algorithms rely on making dynamic choices for scheduling index lookups during query execution in order to prune low-scoring candidate items as early as possible. They typically scan precomputed index lists for text terms or attribute values of structured records in descending order of local (i.e., per-term) scores and aggregate these scores for the same data item into a global score, using a monotonic score aggregation function such as (weighted) summation. Based on clever bookkeeping of score intervals and thresholds for the top-k matches, these index scans can often terminate early, namely as soon as the final top-k results can be safely determined, and thus the algorithm often only has to scan short prefixes of the inverted lists. In contrast to the heuristics adopted by many Web search engines, these threshold algorithms compute exact results and are provably optimal in terms of asymptotic costs.

XML and IR

Efficient evaluation and ranking of XML path conditions is a very fruitful research area. Solutions include various forms of structural joins, multi-predicate merge joins, the staircase join based on index structures with pre- and postorder encodings of elements within document trees [10], and holistic twig joins [6]. The latter, also known as path stack algorithm, is probably the most efficient method for twig queries using a combination of sequential scans over index lists stored on disk and linked stacks in memory. However, these approaches are not dealing with uncertain structure and do not support top-k-style threshold-based early termination.

IR on XML data has become popular in recent years. Some approaches extend traditional keyword-style querying to XML data [7,11], introduced full-fledged XML query languages with rich IR models for ranked retrieval [9,19], or developed extensions of the vector space model for keyword search on XML documents. FleXPath [4] was among the first approaches to combine this theme with full-text conditions over search predicates. Meanwhile, various groups have started adding IR-style keyword conditions to existing XML query languages. TeXQuery is the foundation for the W3C's official full-text extensions to XPath 2.0 and XQuery 1.0. TIX and TAX are query algebras for XML that integrate IR-style query processing into a pipelined query evaluation engine. TAX furthermore comes with an efficient algorithm for computing structural joins. Here, the results of a query are scored subtrees of the data, and TAX already provides a threshold operator that drops candidate results with low scores from the result set. TOSS is an extension of TAX that integrates ontological similarities into the TAX algebra.

XIRQL [9], a pioneer in the field of ranked XML retrieval, presents a path algebra based on XQL, an early ancestor of W3C's XQuery, for processing and optimizing structured queries. It combines Boolean query operators with probabilistically derived weights for ranked result output, thus carrying the probabilistic IR paradigm over to the XML case. Finally, XXL [19], specifies a full-fledged, SQL-oriented query language for ranked XML-IR with a high semantic expressiveness that made it stand apart from the Boolean

XQL and XPath language standards being predominant at that time. For ranked result output, XXL leverages both a standard IR vector space model and an ontology-oriented similarity search for the dynamic relaxation of structure and term conditions. TopX [17], the actual successor of XXL, on the other hand, focuses on a smaller, XPath-like, subset of the XXL query language which allows for a radically different query processing architecture that outperforms XXL in terms of efficiency by a large margin.

Foundations

Applying the TA paradigm for inverted index lists to XML ranked retrieval is not straightforward. In a data-centric XML setting, a ranking of query results to a query is typically induced by defining some form of structural similarity, whereas in a more text-centric view (with a rich mixture of XML tags and text contents), ranking is derived from IR-style text relevance measures, or a combination of structural similarity and text relevance. More precisely, the XML-specific difficulties arise from the following challenges:

Query Processing and Index Structures: Relevant intermediate results to a mixture of structural and content-related search conditions must be tested as to whether they satisfy the path conditions of the query, and this may incur repetitive and expensive *random access* to large, disk-resident index structures. Furthermore, instead of enforcing conjunctive query processing, it is desirable to relax path conditions and rather rank documents by a combination of content scores and an additional degree to which the structural query conditions are satisfied. *Incremental path evaluations* are required when the index structures are accessed mostly using efficient *sequential disk access* in order to limit or entirely avoid the more expensive random accesses. Yet all incremental updates to candidate score bounds during the query processing need to stay monotonic, in order to guarantee a correct algorithmic basis for top-k query evaluation with early candidate pruning.

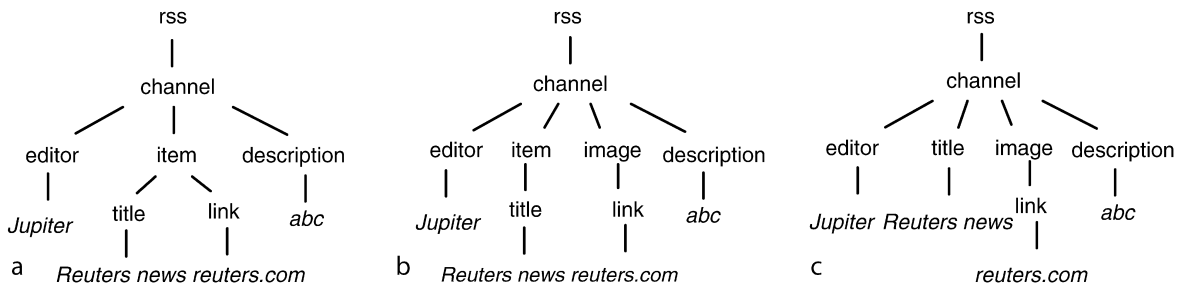
IR Scoring Models and Vague Search: Existing IR scoring models for text documents cannot be directly carried over to the XML case, because they would not consider the specificity of content terms in combination with hierarchical elements or attribute tags. For example, the term "transactions" in a bibliographic data set should be viewed as specific (and lead to a high score) when occurring within elements of type section or caption but be considered less informative within tags like journalname. Furthermore, it should be possible to relax search terms and, in particular, tag names, using tree editing operations, or ontology- and thesaurus-based similarities. For example, a query for a book element about "XML" should also consider a monograph element on "semi-structured data" as a relevant result candidate.

Result Granularity: Scores and index lists refer to individual XML elements and their content terms, but from an IR point-of-view it is desirable to aggregate scores at the document level and return the most relevant XML subtrees, up to the entire XML document, as results. Thus, the query evaluation has to weigh *different result granularities* against each other dynamically in the top-k query processing, and the relevance scoring model should consider XML-specific ranking aspects such as *exhaustiveness* and *specificity* when choosing the most suitable result granularity to address the user's information need.

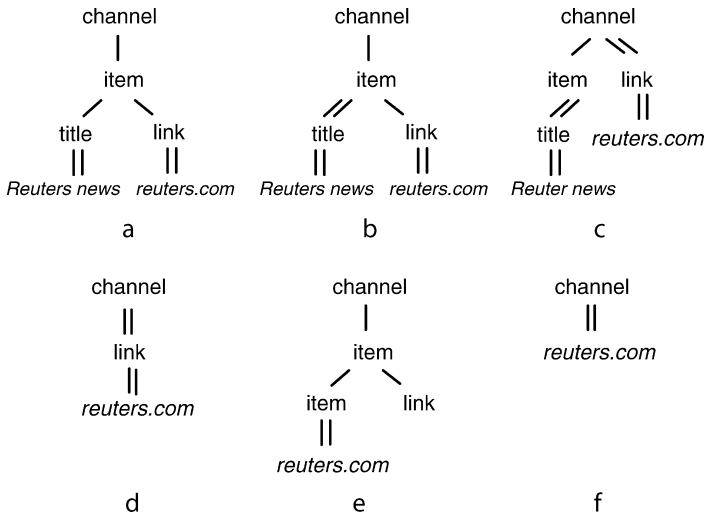
Scoring Structure

Structural similarity is considered in the sense that documents can qualify even if they do not satisfy all path conditions, i.e., if there were too few results otherwise. For dynamically relaxing tag names and structural relationships of tags in path queries, various tree editing operations can be employed, such that only the most similar matches in the collection are returned.

XML data can typically be represented as forests of node-labeled trees. Figure 1 shows a database instance containing fragments of heterogeneous news documents. Figure 2 gives examples of several queries drawn as trees: the root nodes represent the returned answers, single and double edges represent the descendant and child axes, respectively, and node labels stand for names of elements or keywords to be matched. Hence, different queries match the different news documents in Fig. 1. For example, query (a) in Fig. 2 matches document (a) exactly, but would neither match document (b) (since link is not a child of item) nor document (c) (since item is entirely missing). Query (b) matches document (a), also since the only difference between this query and query (a) is the descendant axis between item and title. Query (c) matches both documents (a) and (b) since link is not required to be a child of item while query (d) matches all documents in Fig. 1. Intuitively, it makes sense to return all three news documents as candidate matches, suitably ranked



Ranked XML Processing. Figure 1. Heterogeneous XML database example.



Ranked XML Processing. Figure 2. Query tree patterns and relaxations.

based on their similarity to query (a) in Fig. 2. Queries (b), (c), and (d) in Fig. 2 correspond to *structural relaxations* of the initial query (a) as defined in [4]. In the same manner, none of the three documents in Fig. 1 matches query (e) because none of their title elements contains *reuters.com*. Query (f), on the other hand, is matched by all documents because the scope of *reuters.com* is broader than in query (e). It is thus desirable to return these documents suitably ranked according to their similarity to query (e).

In order to achieve the above goals, [4] defines query relaxations, including *edge generalization* (replacing a child axis with a descendant axis), *leaf deletion* (making a leaf node optional), and *subtree promotion* (moving a subtree from its parent node to its grandparent). These relaxations capture approximate answers but still guarantee that exact matches to the original query continue to be matches to the relaxed query. For example, query (b) can be obtained from query (a) by applying edge relaxation to the axis between `item` and title and still guarantees that documents where title is a child of item are matched. Query (c) is obtained from query (a) by composing edge generalization between item and title and subtree promotion (applied to the subtree rooted at link). Finally, query (d) is obtained from query (c) by applying leaf deletion to the nodes *ReutersNews*, title and item. Query (d) is a relaxation of query (c) which is a relaxation of query (b) which is in turn a relaxation of query (a). Similarly, query (f) in Fig. 2 can be obtained from query (e) by a combination of subtree promotion and leaf deletion. Other works have considered additional query relaxation such as *node renaming*, *node generalization*, *node insertion*, and *node deletion* [1,9,15,16]. The use of schema knowledge can reduce the number of possible relaxed queries by ignoring relaxations that are guaranteed not to lead to additional matches [16].

Amer-Yahia et al. [3] presented strategies to assign scores to query relaxations. These strategies are based on the traditional $tf \cdot idf$ measure derived from IR-style

ranking of keyword queries against an unstructured document collection. The *twig scoring* method introduced in [3] computes the score of an answer taking occurrences of *all* structural and content-related (i.e., keyword) predicates in the query. For example, a match to query (c) would be assigned an *inverse document frequency* score, *idf*, based on the fraction of the number of channel nodes that have a child item with a descendant title containing the keyword *ReutersNews* and a descendant link that contains the keyword *reuters.com*. Such a match would then be assigned a *term frequency* score, *tf*, based on the number of query matches for the specific channel answer.

Scoring Text

A variety of IR-style scoring functions has been proposed and adopted for XML retrieval, ranging from the classic vector space model with its $tf \cdot idf$ family of scoring approaches, typically using *Cosine* measure for score aggregations, over to the theoretically more sound *probabilistic scoring models*, with *Robertson & Sparck-Jones* and *Okapi BM25* being the most widely used ranking approaches in current IR benchmark settings such as TREC or INEX, up to even more elaborated *statistical language models*. An important lesson from text IR is that the influence of the term and document frequency values, *tf* and *df* – in the following referred to as their element-specific counterparts *ftf* and *ef*, should be sub-linearly dampened to avoid a bias for short elements with a high term frequency of a few rare terms. To address these considerations, the TopX engine, presented by Theobald et al. [17,18], adopts the empirically very successful Okapi BM25 probabilistic scoring model to a generic XML setting by computing individual relevance models for each element type occurring in the collection.

For a typical NEXI query pattern of the form $q = //A[\text{about}(.//,t_1,...,t_m)]$, the following *relevance score* is computed for an element e with tag name A:

$$score(e,q) = \sum_{i=1}^{m} \frac{(k_1+1)ftf(t_i,e)}{k + ftf(t_i,e)} \cdot \log\left(\frac{N_A - ef_A(t_i) + 0.5}{ef_A(t_i) + 0.5}\right)$$

$$\text{with } K = k_1\left((1-b) + b\frac{length(e)}{avg\_length_A}\right)$$

Here, $ftf(t_i, e)$ models the *relevance* of a term t_i for an element's *full content*, i.e., the frequency of t_i in all the descending text nodes of element e; while $ef_A(t_i)$ models the *specificity* of t_i for a particular element with tag name A by capturing how many times t_i occurs under a tag A across the whole collection having N_A elements with this tag name.

That is, this extended BM25 model computes a separate relevance model for each term t_i with respect to its enclosing tag name A, thus maintaining detailed element frequency statistics $ef_A(t)$ of each individual tag-term pair that occurs in the collection. It provides a smoothed (i.e., dampened) influence of the *ftf* and *ef* components, as well as a compactness-based normalization that takes the average length of each element type into account. Note that the above function also includes the tunable parameters k_1 and b just like the original BM25 model that now even allows for fine-tuning the influence of the *ftf* components and the length normalization for each element type individually – if desired. For an about operator with multiple keyword conditions as used in the NEXI query language of the INEX benchmark series (or similarly for ftcontains in the XPath 2.0 Full-Text specification), that is attached to an element e with tag name A, the aggregated score of e is simply computed as the sum of the element's scores over the individual tag-term conditions. For path queries with more than one structural tag condition or with multiple full-text operators, the content scores of each element can be combined with the structural scores described above.

Various extensions for more specific full-text predicates such as keyword proximity and phrase matching, as well as incorporating ontological concept similarities for query expansion, have been proposed in the literature – some of which are leaving some interesting research questions for a top-k-style query processor, since most of the more sophisticated proximity- or graph-based compactness measures inherently lead to non-monotonic score aggregation functions.

XML Top-k Query Evaluation Techniques

Combining Structure Indexes and Inverted Lists: Kaushik et al. [12] proposed one of the first, exact-match, top-k algorithms for XML by employing various path index operations as basic steps for non-relaxed evaluations of branching path queries. Their strategy combines two forms of auxiliary indexes, a *DataGuide*-like path index for the structure whose extent identifiers are linked to an inverted index for

processing relevance-ranked keyword conditions. The index processing steps are then invoked within a TA-style top-*k* algorithm, involving eager random access to these inverted index structures.

XRank: Among the most prominent IR-related approaches for ranked retrieval of XML data is XRank [11]. It generalizes traditional link analysis algorithms such as *PageRank* for authority ranking in linked Web collections and conceptually treats each XML element as an interlinked node in a large element graph. Then the *element rank* of an XML element corresponds to the authority weight computed over a mixture of containment edges, obtained from the XML tree structure, and hyperlink edges, obtained from the inter-document XLink structure, similar to the HTML case. XRank may indeed return deeply nested elements but merely supports conjunctive keyword search; it does not yet support structured and/or path query languages such as XPath. For efficient retrieval of multi-keyword queries, it also uses inverted lists sorted in descending order of element ranks and sketches the usage of standard threshold algorithms for pruning the search space.

FlexPath: FlexPath [4] integrates structure and keyword queries and regards the query structure as templates for the context of a full-text keyword search. The query structure (as well as the content conditions) can be dynamically relaxed for ranked result output according to predefined tree editing operations when matched against the structure of the XML input documents. The FlexPath query processor already comprises the usage of top-*k*-style query evaluations for a slightly modified, XPath-like, query language that later evolved as part of the official W3C Full-Text extensions to XPath 2.0 and XQuery 1.0. Like [12], it uses separate index structures for storing and retrieving the structural and content-related conditions of an XPath 2.0 Full-Text query; it may thus require a substantial amount of random access to disk-resident index structures for resolving the final structure of a result candidate.

Whirlpool: The Whirlpool system introduced by Marian et al. [14] provides a flexible architecture for processing top-*k* queries on XML documents *adaptively*. Whirlpool allows partial matches to the same query to follow different execution plans, and takes advantage of the top-*k* query model to make dynamic choices during query processing. The key features of Whirlpool are: (i) a partial match that is highly likely to end up in the top-*k* set is processed in a prioritized manner, and (ii) a partial match unlikely to be in the top-*k* set follows the cheapest plan that enables its early pruning. Whirlpool provides several adaptivity policies and supports parallel evaluation; details on experimental results can be found in [14].

TopX: The biggest challenge in further accelerating full-text query evaluations over large, semi-structured data collections lies in finding appropriate encodings of the XML data, for indexes that can be read *sequentially* in big chunks directly from disk when the collection (or index) no longer fits into the main memory of current machines. Thus, the TopX engine [17,18] operates over a *combined inverted index* for content- and structure-related query conditions by precomputing and materializing joins over tag-term pairs, the most common query patterns in full-text search. This simple precomputation step makes the query processing more scalable, with an encoding of the index structure that is easily serializable and can directly be stored sequentially on disk just like any inverted index, for example using conventional B^+-tree indexes or inverted files.

At query processing time, TopX scans the inverted lists for each tag-term pair in the query in an interleaved manner, thus fetching large element blocks into memory using only sorted access to these lists and then iteratively joining these blocks with element blocks previously seen at different query dimensions for the same document. Using pre-/postorder tree encodings [10] for the structure, TopX only needs a few final random accesses for the potential top-*k* items to resolve their complete structural similarity to a path query. An extended hybrid indexing approach using a combination of DataGuide-like path indexes and pre-/postorder-based range indexes can even fully eliminate the need for these random accesses – however at the cost of more disk space.

TopX further introduces pluggable extensions for *probabilistic candidate pruning*, as well as a *probabilistic cost-model* for adaptively scheduling the sorted and random accesses, that help to significantly accelerate query evaluations in the presence of additional, precomputed index list statistics such as index list selectivities, score distribution histograms or parameterized score estimators, and even index list (i.e., keyword) correlations. For *dynamic query expansions* of tag and term conditions, TopX can incrementally merge the inverted lists for similar conditions obtained from an exchangeable background thesaurus such as *WordNet* or *OpenCyc*. Thus, TopX provides a whole toolkit of

specialized top-*k* operators for efficient full-text search, including *incremental merge operators* for dynamic query expansion and *nested top-k operators* for high-dimensional phrase expansions.

Key Applications

Scalable, Web-Style Search over Semi-structured Collections: Efficient IR over large Web collections will remain one of the most challenging applications for XML-top-*k* query processing with full-text search, with an ever-increasing demand for scalability, interactive runtimes, and vague search involving dynamic query relaxation and/or expansion over heterogeneous collections or unknown schemata.

INEX Benchmark Series: INEX provides a comprehensive forum for IR research on semi-structured data, that goes beyond using the formerly prevalent synthetic data collections such as XMark or XBench for evaluating retrieval quality in true IR-style settings, with a variety of subtasks, XML-IR-specific evaluation metrics, and peer assessments of retrieval results [5].

Future Directions

Graph Top-k: Current XML-top-*k* algorithms are restricted to XML data trees. Future work could focus on further generalizing the scoring approach, in order to handle cycles arising from inter- or intra-document XLinks, with the need to still derive tight and accurate bounds for early candidate pruning. This may involve efficient index structures for arbitrary graphs and potentially non-monotonic score aggregation functions to incorporate graph compactness measures such as *Steiner trees*.

More XQuery: Similarly, current work has only focused on implementing various subsets of the XPath query language. Providing top-*k*-style bounds and pruning thresholds for more complex XQuery constructs such as *loops* and *if-cases* would be an intriguing issue for future work.

Experimental Results

Extensive experiments can be gleaned from the various approaches presented in the literature, see, e.g., [3,4,12,14,17].

Data Sets

Links to the INEX IEEE and Wikipedia collections can be obtained from the INEX homepage: http://inex.is.informatik.uni-duisburg.de

URL to Code

http://topx.sourceforge.net

Cross-references

- ▶ Text Indexing and Retrieval
- ▶ XML Information Integration
- ▶ XQuery Full-Text

Recommended Reading

1. Amer-Yahia S., Cho S., and Srivastava D. Tree pattern relaxation. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002. pp. 496–513.
2. Amer-Yahia S., Curtmola E., and Deutsch A. Flexible and efficient XML search with complex full-text predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 575–586.
3. Amer-Yahia S., Koudas N., Marian A., Srivastava D., and Toman D. Structure and content scoring for XML. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005.
4. Amer-Yahia S., Lakshmanan L.V.S., and Pandit S. FleXPath: flexible structure and full-text querying for XML. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 83–94.
5. Amer-Yahia S. and Lalmas M. XML search: languages, INEX and scoring. ACM SIGMOD Rec., 35(4):16–23, 2006.
6. 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.
7. Cohen S., Mamou J., Kanza Y., and Sagiv Y. XSEarch: a semantic search engine for XML. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 45–56.
8. Fagin R., Lotem A., and Naor M. Optimal aggregation algorithms for middleware. J. Comput. Syst. Sci., 66(4):614–656, 2003.
9. Fuhr N. and Großjohann K. XIRQL: a query language for information retrieval in XML documents. In Proc. 24th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2001. pp. 172–180
10. Grust T., van Keulen M., and Teubner J. Staircase join: teach a relational DBMS to watch its (axis) steps. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 524–525.
11. Guo L., Shao F., Botev C., and Shanmugasundaram J. XRank: ranked keyword search over XML documents. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003.
12. Kaushik R., Krishnamurthy R., Naughton J.F., and Ramakrishnan R. On the integration of structure indexes and inverted lists. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004.
13. Kilpeläinen P. and Mannila H. Retrieval from hierarchical texts by partial patterns. In Proc. 16th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1993, pp. 214–222.
14. Marian A., Amer-Yahia S., Koudas N., and Srivastava D. Adaptive processing of top-k queries in XML. In Proc. 21st Int. Conf. on Data Engineering, 2005, pp. 162–173.
15. Schenkel R., Theobald A., and Weikum G. Semantic similarity search on semistructured data with the XXL search engine. Inf. Retr., 8(4):521–545, 2005.

16. Schlieder T. Schema-driven evaluation of approximate tree-pattern queries. In Advances in Database Technology, Proc. 8th Int. Conf. on Extending Database Technology, 2002, pp. 514–532.
17. Theobald M., Schenkel R., and Weikum G. An efficient and versatile query engine for TopX search. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005.
18. Theobald M., Schenkel R., and Weikum G. The TopX DB&IR engine. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2007, pp. 1141–1143.
19. Theobald A. and Weikum G. Adding relevance to XML. In Proc. 3rd Int. Workshop on the World Wide Web and Databases, 2000, pp. 105–124.

Ranking

► Web Search Relevance Ranking

Raster Data Management

► Storage of Large Scale Multidimensional Data

Raster Data Management and Multi-Dimensional Arrays

Peter Baumann
Jacobs University, Bremen, Germany

Synonyms

Array databases; Raster databases

Definition

In this enry, the management of raster data is described based on the concept of an array database system. An array database system is a database system that supports the array (also called raster) data structure, as understood in programming languages: a homogeneous collection of data items where each item has a coordinate associated, and coordinates sit at grid points in a rectangular, axis-parallel subset of the Euclidean space $\mathbf{Z}^d$ for some $d > 0$.

Such arrays convey a very regular structure with a well-defined neighborhood relation between its cells, which requires suitable storage structures. Frequently, operations on cells are applied simultaneously to large contiguous portions of arrays, such as overlaying two equally-sized images; efficient query processing needs to provide support for such patterns. Finally, arrays tend to be very large, with single objects frequently ranging into Terabyte and soon Petabyte sizes; for example, today's earth and space observation archives grow by Terabytes a day.

Query language support for arrays consists of array primitives which can be nested as in standard SQL, and preferably integrated into SQL. Once sufficiently declarative query languages are available, query optimization and parallelization can be applied which has been proven to accelerate array evaluation up to orders of magnitude. Storage management must give efficient access to large arrays and sub-arrays thereof. Due to the large sizes, compression often is applied, but some high-volume applications still require storage hierarchies.

Today, an important application domain is Web services on geo raster data, such as 1-D environmental sensor time series, 2-D satellite images, 3-D x/y/t image time series (Fig. 1), 3-D x/y/z exploration data, and 4-D x/y/z/t climate simulation data. Other spatio-temporal applications can be found, e.g., in the Life Sciences. *On-line analytical processing* (OLAP) and Statistical Databases consider arrays as well, often combining the time dimension with abstract axes such as sales and products.

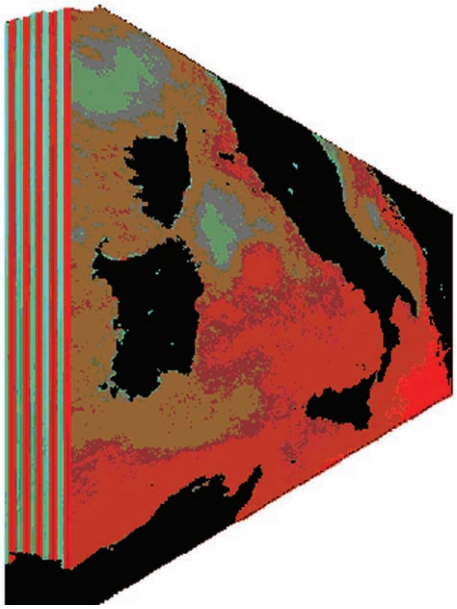
Historical Background

The array paradigm is not directly supported by the relational data model. Therefore, two different approaches have traditionally been pursued to achieve array support: BLOBs and OLAP data *cubes*.

Imagery in Array Databases

If stored in databases, image pixel matrices go into BLOBs (Binary Large OBjects), linearized and encoded in some data exchange format, and result in a byte string whose semantics is unknown to the database system. The database system, therefore, cannot provide any array operation aside from reading and writing the complete BLOB. As a consequence, for many potential application domains, users believe that databases cannot support their raster data, and they still tend to develop their own file-based services in an ad-hoc manner.

The requirement for further array processing capabilities distinguishes array databases from multimedia databases and imaging systems. Array databases are different from multimedia databases which analyze images using some built-in algorithms to obtain



Raster Data Management and Multi-Dimensional Arrays. Figure 1. Retrieval result from a 3-D x/y/t satellite image time series data cube on sea surface temperature; original data cube contains about 10,000 satellite images (image: rasdaman screenshot, data: NASA/DLR).

feature vectors. Subsequently, search is performed on the feature vectors, not the images. Array databases, conversely, do not attempt to interpret the raster data, but always operate on the cell values themselves.

Likewise, array databases are different from image processing and understanding systems. Imaging systems offer elaborate functionality, however are not constrained to excessively large images in relation to main memory size. Array databases, conversely, constrain themselves in functionality to remain safe in evaluation, but aim to have no object size limit.

Statistics Data in Array Databases

Statistical data are usually modeled as OLAP data cubes. Today, the main paradigm is Relational OLAP (ROLAP) where each data item, together with its coordinates, is stored in a so-called fact table in a relational DBMS. This explains why ROLAP is suitable only for very sparse data, where listing only valid points together with their coordinates represents an efficient compression scheme.

Relational query operators, extended with specific data cube operators, provide powerful analysis capabilities on such data cubes.

A Unified View

Although treated differently, both images (here understood to have any spatio-temporal dimensions, not just 2-D x/y axes) and data cubes share characteristics to a large extent. Both consist of a data structure mapping n-D coordinates with rectangular boundaries into a value space. Operations also convey similarities, as the following examples show: Subsetting (cutting out sub-cubes) is known in both worlds, and an OLAP roll-up from days to weeks is equivalent to scaling an image by a factor of seven using linear interpolation.

Actually the main difference motivating different treatment lies in a data property. Statistical data tend to be very sparse, typically only up to 5% of the data cube holds values of interest. Image data, on the other hand, tend to be 100% dense.

Some authors [7] argue that OLAP data cubes are not strictly arrays because, in the case of categorical axes (in OLAP called measures), the neighborhood relation between points does not induce a strict ordering. Others [2] subsume data cubes under the array paradigm, for example because categorical measures ultimately are mapped to integer coordinates during physical modeling.

History and Current State

A first important step beyond BLOBs was accomplished with PICDMS [4] where a 2-D array query language, still procedural and without suitable storage support, has been introduced. A declarative n-D array query language with an algebraic foundation and a suitable architecture, which is implemented and in operational use, has been presented in [2,1]. Marathe and Salem present a 2-D database array language [8]. In [4] nested relational calculus is extended with multidimensional arrays, obtaining a model called NCRA and a query language derived from it, AQL. Both seem to be more theoretically motivated and aim in particular at complexity studies for array addressing. Mennis et al. [9] introduce 3-D map algebra, a framework suitable for handling 2-D and 3-D geo raster data.

While OLAP is a well established business today, arrays are considered by industry only recently. ESRI's ArcSDE and Oracle's GeoRaster cartridge offer tiled storage of 2-D maps with mainly spatial subsetting support; the rasdaman system offers n-D full query support.

Foundations

The collection paradigm in databases encompasses sets, bags, lists, and arrays. The array concept forms a

separate, distinct information category. While it might be emulated, e.g., by nested lists, this will not lead to usable query concepts (e.g., for slicing a data cube along all its axes), nor can this be a basis for efficient implementations.

Consequently, all database aspects need to be reconsidered for array support, including conceptual modeling (what are appropriate operations?), storage management (how to manage objects spanning many disk blocks, if not several media?), and query evaluation (what are efficient and optimizable processing strategies?).

Conceptual Modeling

Formally, an array a is a function a: $X \rightarrow F$ where X is a finite axis-parallel hypercube in Euclidean space $\mathbf{Z}^d$ for some $d > 0$ and F is some algebra. Let X be the array's domain, F its cell type, and the individual locations $x \in X$ carrying some value $f \in F$ the array's cells.

Following good practice in databases, an array query language should be declarative and safe in evaluation. Declarative in this context means that there is no explicit iteration sequence over an array (or part of it) during evaluation – conceptually, all cells should be inspected simultaneously. This opens up avenues for efficient storage and evaluation patterns, and query optimization in general (see below). Avoiding explicit iterations also contributes to be safe in evaluation, i.e., every query is evaluated in a finite number of (finite-time) steps.

Actually it turns out that a wide range of practically relevant operations can be expressed using only a few primitives [2]. The MARRAY operator creates an array over some given domain and sets its cells by evaluating an also given expression at each cell location. An application of the MARRAY operator has the general form

marray *index-range-specification*
values *cell-value-expression*

where *index-range-specification* defines the resulting array's domain and binds an iteration variable to it. In *cell-value-expression* the value of the cells are determined for each cell coordinate; this expression may or may not use the current value of the iteration variable, e.g., to read out cells from some given array. The first example, domain subsetting, makes use of this addressing by copying the values of cells within the cutout region from the original array into the new array.

Example: "A cutout of array a specified by the corner points (100,100) and (200,300)."

marray p in [100:200,100:300]
values $a[p]$

In the query language this particular application pattern of MARRAY can be abbreviated as

a[100:200,100:300]

Aside from copying cells as above, the resulting array can also have new values assigned which may or may not depend on other arrays.

Example: "Array a (which is known to be of size 1024×768), with intensity reduced by a factor of 2." Auxiliary function dom(a) returns the domain of a, over which the query has to iterate:

marray p in dom(a)
values $a[p]/2$

Such operations are abbreviated in the query language by applying the operation on hand directly to the array, in the example obtaining:

$a/2$

All unary and binary operations on cells can thus be lifted to become array operations.

Example: "Array a's values where they are above threshold t, and 0 otherwise."

$(a > t) * a$

In this example, Boolean and arithmetic operations are combined. Similar to many programming languages, Boolean results from the comparison are interpreted as 0 and 1 for the subsequent multiplication.

The CONDENSE operator aggregates cell values into one scalar result, similar to SQL aggregates. Its application has the general form

condense *condense-op*
over *index-range-specification*
using *cell-value-expression*

where, like with MARRAY before, *index-range-specification* indicates the array index domain to be iterated over and binds a variable to it. Similarly, *cell-value-expression* represents an expression to be evaluated for each index range coordinate. The additional element, *condense-op*, specifies the aggregating operation used to combine the cell value expressions into one single value.

Example: "The sum of all values in a."

condense+
over p in sdom(a)
using $a[p]$

This is abbreviated as *add_cells*(a); further predefined condensers include *count_cells*(), minimum, maximum, and quantifiers.

Histogram computation demonstrates nesting of these operations.

Example: "A histogram of 256 buckets over 8-bit greyscale array a."

marray n in [0:255]
values count_cells($a = n$)

The inner expression, $a = n$, is an application of the formerly introduced "lifted" operation. For each coordinate p of a, the comparison $a[p] = n$ is performed, yielding a Boolean array.

The SORT operator allows slicing of an array along one of its axes and reordering of slices according to some sorting criterion.

By embedding this into a set-oriented model with array-valued attributes one can write queries over sets of arrays. The next example assumes a table *BrainActivationMaps* where one of the attributes, named *data*, is an array. The data array might contain processed scans of human brains indicating activity like electrical flow, temperature, blood pressure, etc. Note that for the query it does not matter whether the *data* array is, say, 2-D or 3-D. This way queries can be kept rather generic.

Example: "Tuple identifier and histogram of all those *BrainActivationMaps* tuples where the average data intensity exceeds threshold 127."

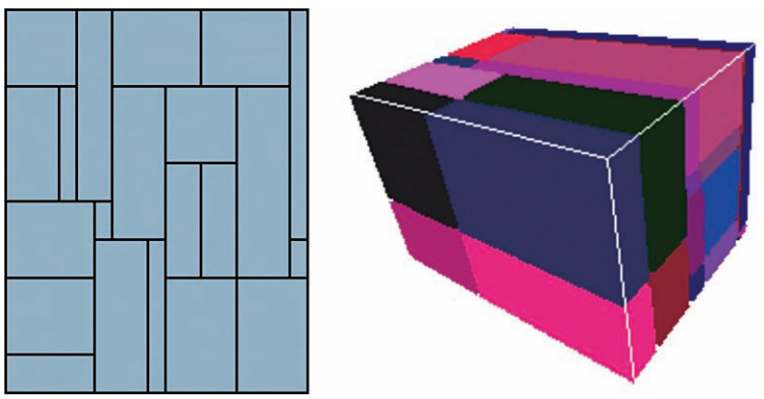
select *id*,
marray n in [0:255]
values *count_cells*(*bam.data* = n)
from *BrainActivationMaps* as *bam*
where *avg_cells*(*bam.data*) > 127

Such languages allow formulating statistical and imaging operations which can be expressed analytically without using loops. In [6] it has been proven that the expressive power of such array languages in principle is equivalent to relational query languages with ranking.

Physical Modeling

In almost all practically relevant scenarios, array objects are by orders of magnitude larger than disk blocks. The foremost task for an array storage manager is to preserve spatial proximity on disk. For n-D data this obviously is impossible to achieve on a linear storage space, hence approximations need to be employed. One such technique is tiling, adopted from imaging. Tiling (also called chunking) partitions an n-D cube into a set of non-overlapping n-D subcubes (Fig. 2), each of which is stored in a BLOB in the traditional manner. The partitioning pattern can be chosen freely, which opens up a wide space for storage optimization. In the most simple case, equally sized tiles are employed where the tile size is chosen suitably for the disk and bus specs; an advanced alternative is to determine an optimal tile distribution with individual tile proportions and sizes for a given query set.

A spatial index, such as the R-Tree, helps to quickly determine the tiles affected by a query, which usually is a *range query*. As opposed to spatial (i.e., vector) databases the situation is simple: the target objects, which



Raster Data Management and Multi-Dimensional Arrays. Figure 2. Sample tiling of 2-D and 3-D arrays.

have a regular box structure, partition a space of known extent. Hence, almost any spatial index will perform decently.

Often compression of tiles is advantageous. Still, in face of very large array databases tertiary storage may be required, such as tape robots [10,11].

Query Evaluation

A tile-based storage structure suggests a tile-by-tile processing strategy. Indeed a large class of practically relevant queries can be evaluated by inspecting a tile stream, rather than loading the whole source object into main memory. Additionally, for some query types such as condensers, tile streaming can be terminated prematurely.

It turns out that array query evaluation times typically are CPU bound, except for the very rare cases where only domain subsetting and no processing is required. The reason lies in the large number of array cells to be processed per query. Preliminary results show that query parallelization by assigning tiles to different CPUs yields promising performance gains [5].

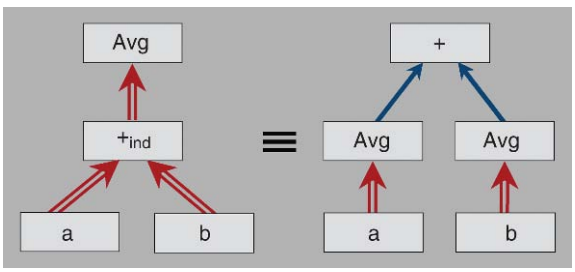
Query Optimization

Array queries lend themselves well towards *query optimization*. Currently only heuristic optimization is reasonably understood, but shows good results. Consider the query

select *avg_cells*($a + b$)
from *a*, *b*

Figure 3 shows an equivalence rule which can speed up evaluation: substituting the left-hand side array expression with the right-hand side yields

select *avg_cells*(a) + *avg_cells*(b)
from *a*, *b*



Raster Data Management and Multi-Dimensional Arrays. Figure 3. Algebraic equivalence rule "*avg_cells*(*m1* + *m2*) ≡ *avg_cells*(*m1*) + *avg_cells*(*m2*)."

This query involves two array traversals and two (negligible) scalar operations. Bottom line, three array traversals have been reduced to two this way.

Key Applications

In many cases where some phenomenon is sampled or simulated, the result is a rasterized data set. Given the high volumes and multi-faceted retrieval arising, there is a remarkably wide application domain for array databases. Briefly, it can be summarized as sensor, image, and statistics data in the widest sense. Exemplarily geo and life sciences will be inspected next.

Earth Sciences

By far, the most important and visible application domain for large-scale array information systems is geospatial raster data, with high public visibility through novel, user-friendly interaction techniques like the Virtual Globes of NASA WorldWind (worldwind.arc.nasa.gov) and GoogleEarth (earth.google.com).

The Open GeoSpatial Consortium (OGC, www.opengeospatial.org), in collaboration with ISO, OASIS, W3C and others, provides standards for open, interoperable geo Web service interfaces. OGC develops a family of modular geo service standards, of which the Web Coverage Service (WCS) suite is particularly relevant for raster data (there also called coverage data). It offers basic subsetting services, scaling, and reprojection. This static request type is augmented with a coverage processing language in the Web Coverage Processing Service (WCPS) Implementation Specification [3].

A typical WCPS request computes the Normalized Difference Vegetation Index (NDVI), which computes a measure of the degree of vegetation for each pixel. For a multi-spectral satellite image *s* with near-infrared channel *nir* and red channel *red*, the NDVI is defined as:

NDVI(s) = (*s.nir* - *s.red*)/(*s.nir* + *s.red*)

The result is a real value between -1 and +1; the closer it is to +1, the higher is the likelihood for vegetation. For some server object *LandsatScene* (note that WCPS lists single objects and does not operate on sets like database languages) an 8-bit greyscale NDVI image is specified as follows:

for *s* in (*LandsatScene*)
return
encode((char) (255 * (*s.nir* - *s.red*)/(*s.nir* + *s.red*) + 1), "TIFF")

Life Science

Human Brain Imaging In human brain imaging, the research goal is to understand the relations between brain structure and its function. In experiments, human subjects have to perform some mental task while activity parameters such as brain temperature, electrical activity, and oxygen consumption are measured by PET or fMRI CAT scans. The resulting 3-D x/y/z data cubes have a resolution of 1 mm and an overall volume in the Megabyte size. In a computationally expensive warping operation, the brain images get normalized against some chosen standard brain so that organs always sit at known voxel coordinates (Fig. 4 left). In the end, a brain image set as large as possible is desired to achieve high statistical significance.

Traditionally, feature search is the only property through standard SQL on structured and semi-structured data; specialized tools can perform search on single images or small sets thereof. With array databases, thousands of experiments each with large image sets can be searched by brain feature. The brain organs can be registered as voxel masks (Fig. 4 right). The query types arising mainly perform standard statistics per brain.

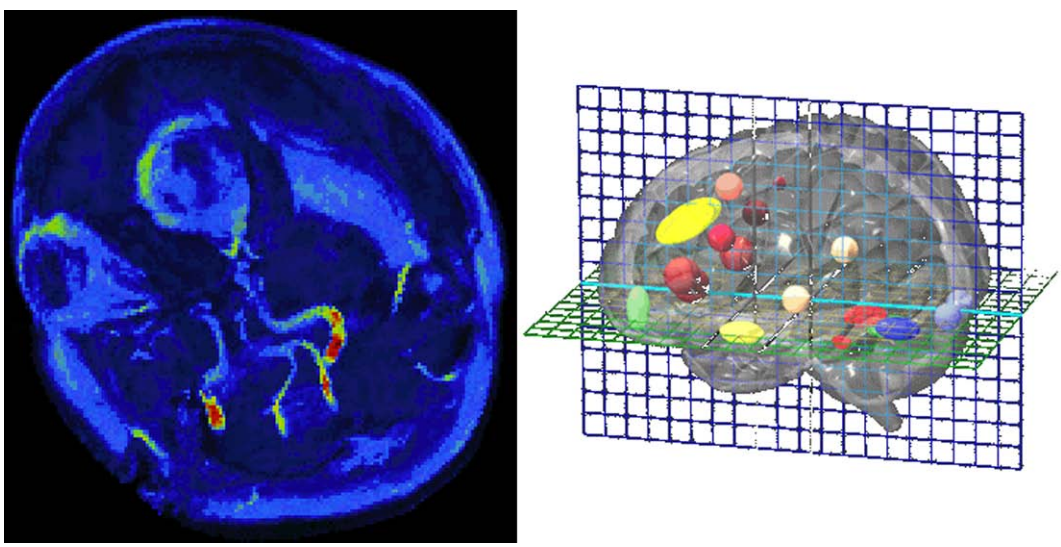
Example: "A parasagittal view of all scans containing critical Hippocampus activations, TIFF-coded." Positional parameter \$1 denotes the slicing position, \$2 the intensity threshold value, \$3 the confidence, as chosen by a user through some browser interface.

```
select tiff(ht[$1, *:*, *:*])
from HeadTomograms as ht, Hippocampus as mask
where count_cells(ht > $2 and mask)/count_cells
(mask) > $3
```

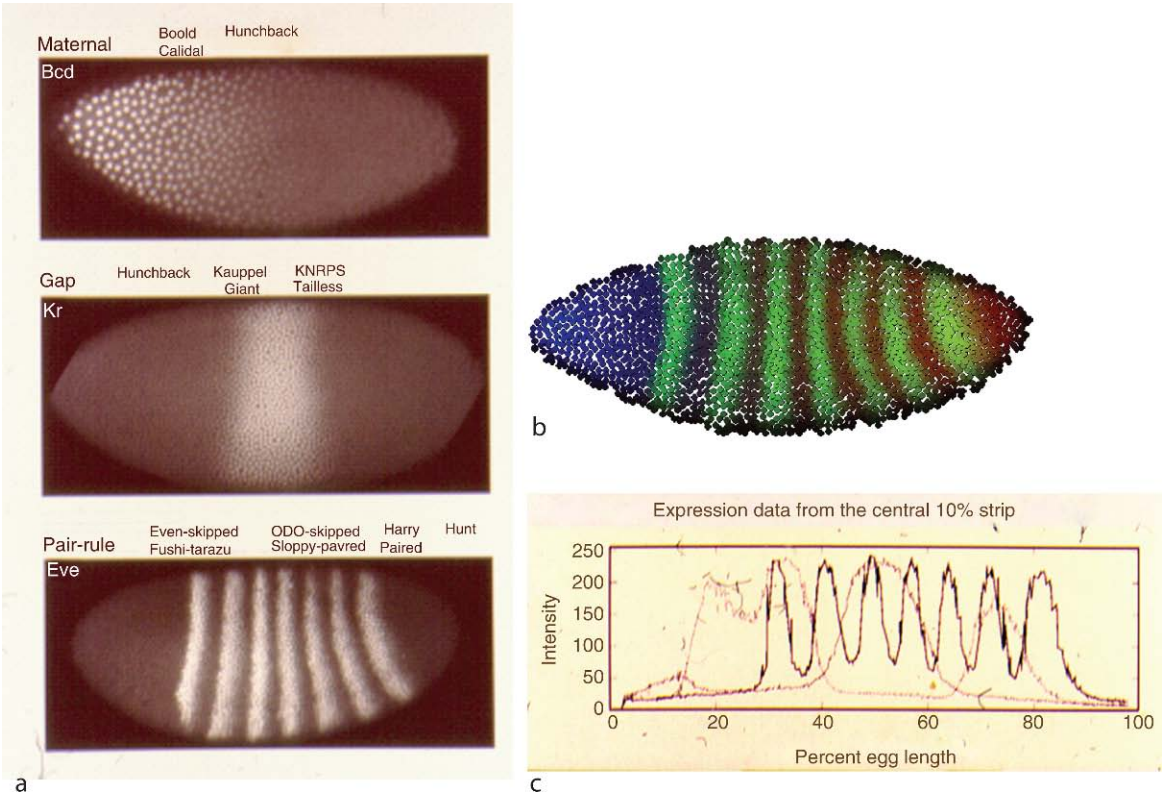
Gene Expression Analysis Gene expression is the activity of reading out genes for reproduction in a living body. The research goal in gene expression analysis is to understand how and when genes express to become manifest in the phenotype. One step towards this is to understand the spatio-temporal expression patterns in a well-known research object, the fruit fly Drosophila melanogaster.

The staining process delivers activity patterns (Fig. 5 left). It is common to combine three gene images into one by randomly assigning them to the red, green, and blue channel, resp. (Fig. 5 top right). Traditionally, diagrams like Fig. 5 bottom right are constructed.

Array queries allow searching the resulting 4-D x/y/z/t activity cube and generate, among others, the views researchers are used to. A gene expression database might contain 4-D objects where the first dimension lists the fruitfly genes in some chosen order and the other three spatial dimensions allow addressing into the fruitfly body. Then, a query can slice these 4-D cubes and recombine these slices into the aforementioned RGB overlay.



Raster Data Management and Multi-Dimensional Arrays. Figure 4. Human brain activation map (left) and brain organ masks (right).



Raster Data Management and Multi-Dimensional Arrays. Figure 5. Gene activity maps for a fruitfly embryo (left), overlay into RGB (top right), slice aggregating activity over the 10% central strip of a snapshot (bottom right).

Example: "Genes $1, $2, and $3 at time $4, as RGB images."

```
select jpeg({1c,0c,0c}*e[$1, *:*, *:*, $4]
+{0c,1c,0c}*e[$2, *:*, *:*, $4]
+{0c,0c,1c}*e[$3, *:*, *:*, $4])
from EmbryoImages as e
where oid(e) = 193537
```

Future Directions

Array databases are in their infancy, with many inviting research avenues. On a conceptual level, a unified algebraic array theory seems promising which is to unify (dense) image handling with (sparse) statistical and OLAP handling. Advanced query optimization deserves further investigation, with topics like cost-based optimization and complex array addressing schemes (such as filter kernels). Physical tuning parameters are not yet fully understood in their effects and interplay. Finally, further experience in as many applications as possible is desirable for a better understanding of the relevant query patterns. The ultimate goal is to establish arrays as first-class database citizens.

Cross-references

- ▶ Biomedical Image Data Types and Processing
- ▶ Digital Elevation Models
- ▶ Discrete Wavelet Transform and Wavelet Synopses
- ▶ Geographic Information System
- ▶ Image Database
- ▶ Index Creation and File Structures
- ▶ On-Line Analytical Processing
- ▶ Query Languages and Evaluation Techniques for Biological SEQUENCE Data
- ▶ Query Load Balancing in Parallel Database Systems
- ▶ Query Optimization
- ▶ Query Processing and Optimization in Object Relational Databases
- ▶ Query Processing in Data Warehouses

► Query Rewriting
► Query Translation
► Range Query
► R-Tree (and Family)
► Scientific Databases
► Semantic Modeling for Geographic Information Systems
► Spatial and Spatio-Temporal Data Models and Languages
► Spatial Data Analysis
► Spatial Data Mining
► Spatial Data Types
► Spatial Indexing Techniques
► Spatio-Temporal Data Warehouses

Recommended Reading

1. Baumann P. On the management of multidimensional discrete data. VLDB J., 4(3):401–444, 1994. Special Issue on Spatial Database Systems.
2. Baumann P. A database array algebra for spatio-temporal data and beyond. In Proc. Fourth Int. Workshop on Next Generation Information Technologies and Systems, 1999, pp. 76–93.
3. Baumann P. and Chulkov G. Web coverage processing service implementation specification. Jacobs University Technical Report #9, July 2007.
4. Chock M., Cardenas A., and Klinger A. Database structure and manipulation capabilities of a picture database management system (PICDMS). IEEE Trans. Pattern Analy. Machine Intell., 6(4):484–492, 1984.
5. Hahn K., Reiner B, Höfling G., and Baumann P. Parallel query support for multidimensional data: inter-object parallelism. In Proc. 13th Int. Conf. on Database and Expert Syst. Appl. 2002, pp. 820–830.
6. Libkin L., Machlin R., and Wong L. A query language for multidimensional arrays: design, implementation and optimization techniques. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 228–239.
7. Machlin R. Index-based multidimensional array queries: safety and equivalence. In Proc. 27th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 2008, pp. 175–184.
8. Marathe A. and Salem K. A language for manipulating arrays. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 46–55.
9. Mennis J., Viger R., and Tomlin C.D. Cubic map algebra functions for spatio-temporal analysis. Cartogr. Geogr. Inf. Sci., 32(1):17–32, 2005.
10. Reiner B. and Hahn K. Hierarchical storage support and management for large-scale multidimensional array database management systems. In Proc. 13th Int. Conf. Database and Expert Syst. Appl., 2002, pp. 689–700.
11. Sarawagi S. and Stonebraker M. Efficient organization of large multidimensional arrays. In Proc. 10th Int. Conf. on Data Engineering, 1994, pp. 328–336.

Raster Databases

► Raster Data Management and Multi-Dimensional Arrays

RBAC

► Role Based Access Control

RBAC Standard

► ANSI/INCITS RBAC Standard

RDF

► Resource Description Framework

Reactive Rules

► ECA Rules

Read/Write Model

► Transaction Models – The Read/Write Approach

Real and Synthetic Test Datasets

Thomas Brinkhoff
Institute for Applied Photogrammetry and Geoinformatics (IAPG), Oldenburg, Germany

Synonyms

Spatio-temporal data generator; Spatio-temporal benchmarking

Definition

In the area of mobile and ubiquitous data management, real and synthetic test datasets are used for experimental investigations of performance and robustness.

Typical applications are the examination of access methods for spatio-temporal databases and the simulation of mobility for location-based services. Besides synthetic and real datasets, combinations of these two types of data are often used that integrate a predefined infrastructure.

Historical Background

Beginning in the mid 1990s, the development of algorithms and data structures for spatio-temporal data took place of the research in the field of pure (geo-) spatial applications. Previous spatial test datasets were not sufficient for investigating the performance and robustness of those algorithms and data structures. Consequently, first data generators for spatio-temporal test datasets were published in the end of the 1990s.

Foundations

Comprehensible performance evaluations are an important requirement in the field of mobile and ubiquitous data management. This demand covers the preparation and use of well-defined test datasets and benchmarks enabling the systematic and comprehensible evaluation and comparison of algorithms and data structures in this area.

In experimental investigations, synthetic data following some statistical distributions as well as data from real-world applications are used as test datasets. The use of *synthetic datasets* allows testing the behavior of an algorithm or of a data structure under exactly specified conditions or in extreme situations. In addition, for testing the scalability, synthetic data sets are often suitable. However, it is difficult to assess the performance of real applications by employing synthetic data. The use of *real datasets* tries to solve this problem. In this case, the selection of the data is crucial. For non-experts it is often difficult to decide whether a dataset reflects a "realistic" situation or not. Furthermore, real datasets are typically connected with a special type of application. For example, a dataset recording cars driving within a city may have completely different properties than the traffic in rural areas or the movement of vehicles in battlefields.

In the area of mobile and ubiquitous data management, *infrastructure-based dataset generators* have become popular. These generators compute moving objects that are restricted by some infrastructure like a network or prohibited areas. The infrastructure may be a real-world dataset or completely artificial. The number, distribution, speed and other properties of the moving objects are influenced by the infrastructure as well as by parameters specified by the user of the generator. Dataset produced by infrastructure-based dataset generators are a compromise between real and synthetic datasets in respect of realism on the one hand and of controllability on the other hand.

Synthetic Datasets

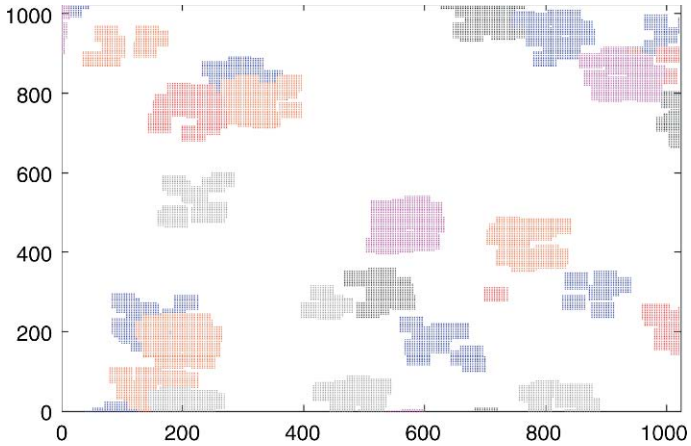
A prominent example for a program providing synthetic spatio-temporal test datasets is the *GSTD (Generate SpatioTemporal Data)* algorithm by Theodorides, Silva and Nascimento [8]. The basic idea of this algorithm is to start with a distribution of point or rectangular objects, e.g., a uniform, a Gaussian or a skewed data distribution. These objects are modified by computing positional and shape changes using parameterized random functions. If a maximum time stamp is exceeded, an object will become invalid. If an object leaves the predefined spatial data space, different approaches can be applied: the position remains unchanged, the position is adjusted to fit into the data space, or the object re-enters the data space at the opposite edge of the data space. In order to create more realistic movements, a later version of the GSTD algorithm considers rectangles for simulating an infrastructure: each moving object has to be outside of these rectangles.

The *Oporto generator* by Saglio and Moreira [7] is designed for a specific scenario: fishing at sea. The generator supports different object types, which allow modeling different behavior. In order to generate smooth movements, objects of one type may be attracted (e.g., fish by plankton) or repulsed by objects of other types (e.g., ships by stormy areas).

G-TERD (Generator for Time-evolving Regional Data) by Tzouramanis, Vassilakopoulos and Manolopoulos [9] generates two-dimensional raster data (see Fig. 1). The user can control the behavior of the generator by defining parameters and statistical models. The parameters influence the color, the maximum speed, size and rotation of the moving regions. Other moving or static objects may have impact on the speed and on the direction of a moving object.

Real Datasets

There exist many spatio-temporal datasets in the WWW [6]. Very often GPS positions of cars or other vehicles like buses, trains or bikes are recorded. An example is the INFATI dataset from the Aalborg University [3]. It represents a collection of spatio-temporal



Real and Synthetic Test Datasets. Figure 1. Raster regions generated by G-TERD.

data that was collected for a project about intelligent speed adaptation. The dataset contains the GPS tracks of about two dozen cars equipped with GPS receivers and logging equipment.

Other popular objects are animals equipped with active satellite tags, e.g., whales, sharks and turtles. The recordings of weather phenomena like hurricanes and the orbits of satellites can also be used as spatio-temporal test datasets.

The section "Data Sets" of this entry gives a small overview on real datasets that are provided for download.

Datasets from Infrastructure-Based Generators

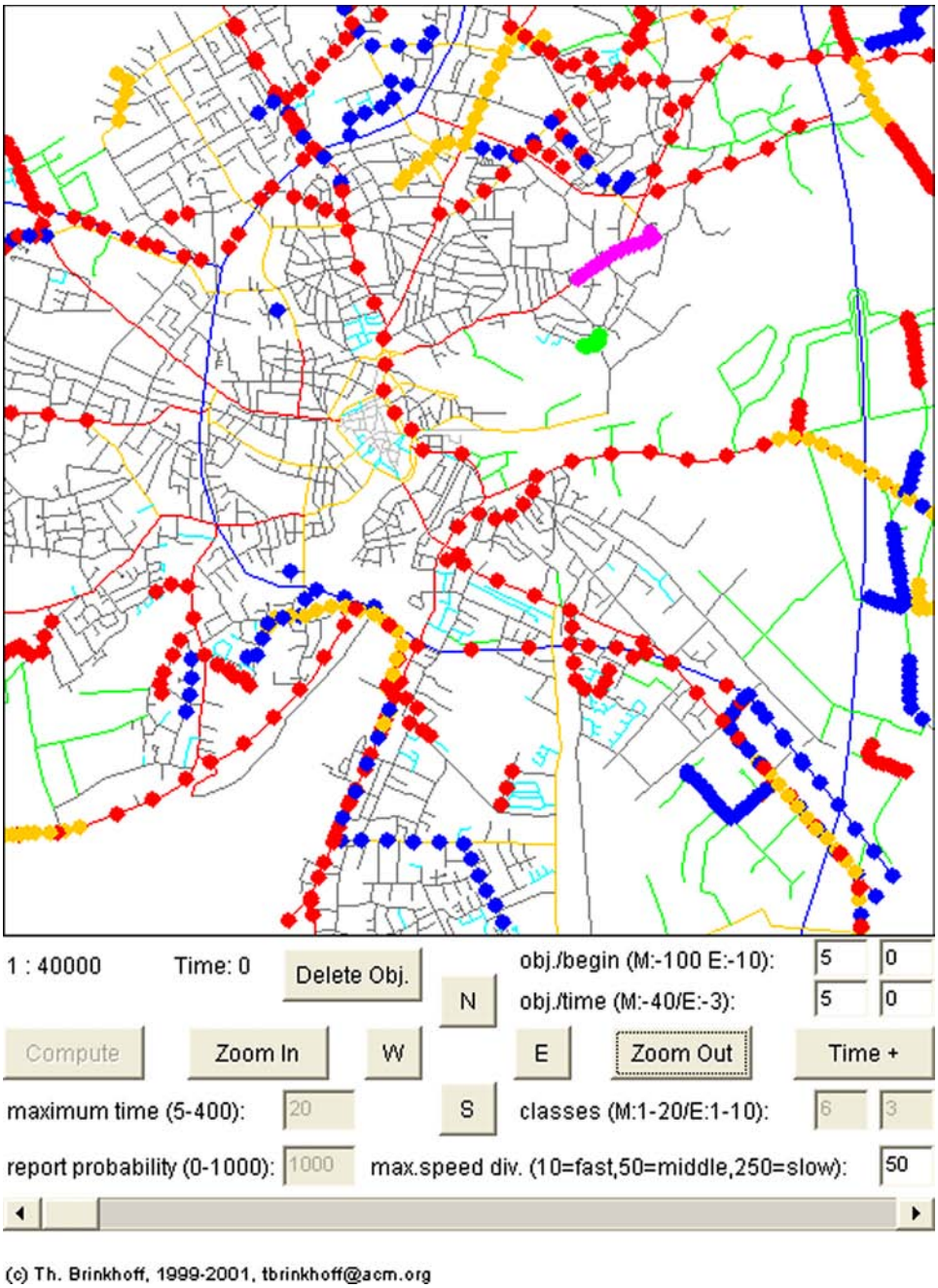
The *Network-based Generator* by Brinkhoff [1] is based on the observation that the motion of objects is often restricted by a network. Examples are streets, railways, air corridors or waterways. Therefore, the generator computes moving objects according to a network that is provided by a file or by a spatial database (see Fig. 2). For each edge of the network, a speed limit and a maximum capacity can be defined. If the number of objects traversing an edge at the same time exceeds the specified capacity, the speed limit on this edge may decrease. In addition, each moving object has a (maximum) speed. The computations of the number of new objects per time stamp, of the start location, of the length of a new route and of the location of the destination are done by time-dependent Java functions that can be overloaded by the user of the generator. This concept allows modeling daily commuting and rush hours. The route of a moving object is computed at the time of its creation. However, the fastest path may change over the time by the motion of other objects and of other influences. Therefore, the re-computation of a route is triggered by events depending on the travel time and on the deviation between the current speed and the expected speed on an edge.

The *City Simulator* by Kaufman, Myllymaki, and Jackson [4] is a scalable, three-dimensional model city that enables the creation of dynamic spatial data simulating the motion of up to one million moving objects. The data space of the city is divided into different types of places that influence the motion of the moving objects: roads, intersections, lawns and buildings are such places that define together a city plan (see Fig. 3). Each building consists of an individual number of floors for modeling the third dimension.

SUMO (Simulation of Urban Mobility) [5] developed by Institute of Transport Research at the German Aerospace Center and the Center for Applied Informatics (ZAIK) is open-source software for traffic simulation. Its objective is to support the traffic research community with a common platform for testing and comparing models of vehicle behavior, traffic light optimization, routing etc. Therefore, the car movement model of SUMO is much more evaluated than the models of the other generators.

Key Applications

The primary field of application of the presented test datasets is the evaluation of spatio-temporal databases, but also in other fields the datasets are used. According to Citeseer and the ACM Portal, applications cover (among others) the evaluation of spatio-temporal data structures, the analysis of spatio-temporal queries



Real and Synthetic Test Datasets. Figure 2. Demo of the network-based generator.

and of mobility patterns, the simulation of wireless environments by mobile agents, the design of (web) server architectures for moving objects, and the test of car-to-car communication.

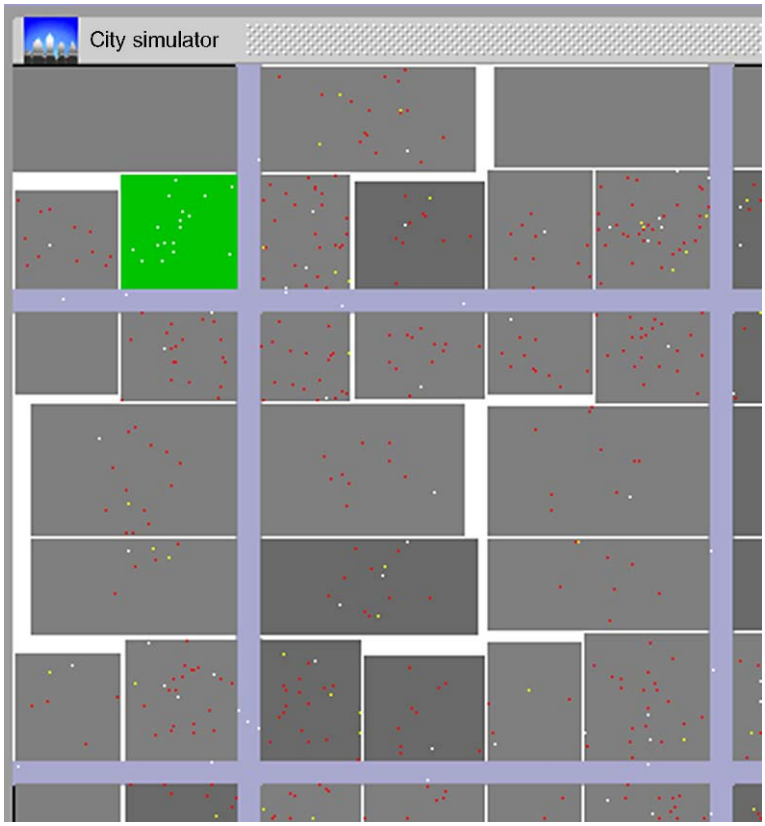
Future Directions

It can be expected that spatio-temporal test datasets will be more often used for evaluations of sensor networks, peer-to-peer communication and positioning techniques.

Data Sets

INFATI Dataset: http://arxiv.org/abs/cs.DB/0410001

Pfoser, D. Where can I get spatio-temporal data? http://dke.cti.gr/people/pfoser/data.html



Real and Synthetic Test Datasets. Figure 3. Visualization of a city plan by the city simulator.

Sea Turtle Migration-Tracking: http://www.cccturtle.org/satellitetracking.php
Unisys Weather – Hurricane/Tropical Data: http://weather.unisys.com/hurricane/index.html
WhaleNet: http://whale.wheelock.edu/whalenet-stuff/stop_cover.html

URL to code

GSTD: see http://www.cs.ualberta.ca/~mn/ for further notices
G-TERD: http://delab.csd.auth.gr/stdbs/g-terd.html
Network-based generator: http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/
Oporto: http://www.inf.enst.fr/~saglio/etudes/oporto/
SUMO: http://sumo.sourceforge.net/

Cross-references

- ▶ Geographic Information System
- ▶ Indexing Historical Spatio-temporal Data
- ▶ Indexing of the Current and Near-Future Positions of Moving Objects
- ▶ Location-based services (LBS)
- ▶ Moving Objects Databases and Tracking
- ▶ Road Networks
- ▶ Spatial and Spatio-temporal Data Models and Languages
- ▶ Spatial Network Databases
- ▶ Spatio-Temporal Trajectories

Recommended Reading

1. Brinkhoff T. A framework for generating network-based moving objects. GeoInformatica, 6(2):153–180, 2002.
2. Jensen C.S (ed.). Special issue on infrastructure for research in spatio-temporal query processing. Bull. Tech. Comm. Data Eng., 26(2):51–55, 2003.
3. Jensen C.S., Lahrmann H., Pakalnis S., and Runge J. The INFATI Data, 2004. Available at: http://oldwww.cs.aau.dk/research/DP/tdb/TimeCenter/TimeCenterPublications/TR-79.pdf
4. Kaufman J., Myllymaki J., and Jackson J. City Simulator. IBM alphaWorks emerging technologies, 2001. Available at: https://secure.alphaworks.ibm.com/aw.nsf/techs/citysimulator
5. Krajzewicz D., Hertkorn G., Rössel C., and Wagner P. SUMO (Simulation of Urban MObility): an open-source traffic simulation. In Proc. of the Fourth Middle East Symp. on Simulation and Modelling, 2002, pp. 183–187.

6. Nascimento M.A., Pfoser D., and Theodoridis Y. Synthetic and real spatiotemporal datasets. Q. Bull. IEEE TC on Data Engineering, 26(2):26–32, 2003.
7. Saglio J.-M. and Oporto M.J. A realistic scenario generator for moving objects. GeoInformatica, 5(1):71–93, 2001.
8. Theodoridis Y., Silva J.R.O., and Nascimento M.A. On the generation of spatiotemporal datasets. In Proc. Int. Symp. on Large Spatial Databases, 1999, pp. 147–164.
9. Tzouramanis T., Vassilakopoulos M., and Manolopoulos Y. On the generation of time-evolving regional data. GeoInformatica, 6(3): 207–231, 2002.

Real-Time Transaction Processing

Jörgen Hansson[1], Ming Xiong[2]
[1]Carnegie Mellon University, Pittsburgh, PA, USA
[2]Bell Labs, Murray Hill, NJ, USA

Synonyms

Time-constrained transaction management

Definition

Real-time transaction processing focuses on (i) enforcing time constraints of transactions, i.e., meet time constraints on invocation and completion, and (ii) ensuring temporal consistency of data, i.e., data should be valid/fresh at the time of usage.

The successful integration of time-cognizant behavior and transaction processing into a database system is generally referred to as a real-time database system (RTDB).

Historical Background

The area of real-time transaction processing has emerged from the need for real-time systems, which often are safety-critical, to handle large amounts of data in a systematic fashion, and the increasing expectation of non-critical applications that have used conventional databases but are now needed to deal with "soft" real-time data applications, e.g., multimedia. Real-time systems have traditionally managed data in an ad hoc manner, i.e., system developers have stored and manipulated data in regular data structures resident in the application code. This approach does not scale well as applications increase in complexity and in their needs for managing large amounts of data. Conventional databases, however, are considered inadequate for handling real-time requirements. Conventional databases impose a throughput-centric design, e.g., maximizing average throughput, and is thus not concerned about the specific outcome of a transaction (in fact, transactions are considered equally important). Furthermore, conventional databases are general-purpose transaction processing system design to be satisfy the needs of a multitude of non-real-time applications. In contrast, a predominant part of the real-time systems are very resource limited, normally these systems have a couple of orders of magnitude less resources, e.g., primary memory, and are not using hard disks. Thus, system functionality need to be tailored to application-specific needs to accommodate a minimum footprint and overcome different architectural assumptions and data requirements.

Note that the term real-time has many connotations in industry and in the general literature it is often used interchangeably to denote that something is fast. This differs from the notion used here, where real-time is synonymous to the notion of predictability. The notion of real-time indicates that the correctness of a result depends not only on the logical result of its computation but also on the time at which the result is derived. Thus, the system correctness depends both on the functional and temporal behavior of the system execution. The temporal behavior is tightly connected to the time constraints associated with transactions and data, e.g., deadlines. This requires that a transaction management system needs to enforce that the timeliness of transactions and data, in turn requiring that scheduling and concurrency control algorithms are time-cognizant.

Examples of real-time applications are control systems, multimedia, to air-traffic control, and as evident, these applications differ in their real-time requirements, complexity, and the type of data they manage. In order to maintain external/internal correctness, real-time systems have to respond to input stimuli and by an actuator, i.e., produce an output result/action within a finite and sufficiently small time bound. These systems must feature predictability, i.e., the ability to show that the system meets the specified requirements under various conditions the system is expected work under [11].

Several research platforms have developed, e.g., ARTS-RTDB & BeeHive (both from University of Virginia, USA), COMET (Linköping University, Sweden), DeeDS (University of Skövde, Sweden), REACH (Technical University of Darmstadt, Germany), Rodain

(University of Helsinki, Finland), and STRIP (Stanford University, USA).

Foundations

Real-transaction processing systems operate under and are benchmarked against different performance goals, correctness criteria, and assumptions applications than throughput-centric systems. The performance metrics adopted for real-time transaction processing system reflect that real-time transactions have time constraints on the execution, and the validity of the data. These requirements impose constraints on the system, which is reflected in the performance metrics used for measuring the timeliness of the system, in addition to conventional metrics for measuring consistency. Typical performance metrics, out of which several originate from the area of real-time systems, include deadline miss ratio of transactions, tardiness/lateness of transactions, cost/effects due transactions missing their deadlines, data temporal consistency (external and logical). The requirements of timeliness can exceed that of consistency and isolation, i.e., it might be more important to deliver a result of adequate precision/quality in time than delivering an exact result late. This implies that correctness of a result can be traded for timeliness by relaxing consistency (referring to ACID properties).

Transaction Model

Real-time transactions are typically characterized along the dimensions of the temporal scope, criticality, and transaction type.

The temporal scope of a transaction is typically described by an arrival time a_i, a release time r_i, a (worst case) execution time e_i, and a deadline d_i, where the following conditions hold (the times are expressed as absolute variables): $a_i \leq r_i < d_i$, $r_i + e_i \leq d_i$. A system is, thus, considered schedulable if it can be shown that all transactions can meet their deadlines.

Real-time transactions can be categorized given the criticality and the stringency of meeting their deadlines and the notions of hard, firm, and soft deadlines are generally used. A hard deadline is critical and, thus, the transaction must always complete within the deadline. Missing a hard deadline will have severe or even cataclysmal consequences. In contrast, firm and soft deadlines are used to for transactions when it system can tolerate occasional time constraint violations. The distinction between them is in the utility of completing a task after its deadline; a soft deadline indicates there is often value of completing the task albeit late. A firm deadline indicates that there is no value for late completion and tardy transactions, i.e., tasks running beyond their deadline, can thus be aborted to minimize resource usage.

Transactions may have precedence constraints that put constraints on the relative order of execution among transactions.

The temporal consistency of a data object has two parts: *absolute consistency*, which reflects the state of an external environment and how that state is reflected in the database, and *relative consistency,* which concerns the consistency among data elements used to derive other data. Thus, absolute consistency (a.k.a. external consistency) is necessary to ensure that a system's view of the external environment (e.g., the controlled system) is consistent with the actual state of the environment. Relative consistency (a.k.a. logical consistency) ensures that only valid data is used to derive new data. To express temporal consistency, a real-time data object o_j is annotated with an absolute validity interval avi_j denoting the time length and a timestamp ts_j when the data object was last updated/sampled. The data object is considered valid in the time interval $[\mathrm{ts}_j, \mathrm{ts}_j + \mathrm{avi}_j]$.

To define the notion of a relative validity interval *rvi,* a *relative consistency set R* is introduced for each derived data object, which contains the set of data objects used for its derivation. Furthermore, each such set is has a relative validity interval denoted R_{rvi}. A data object o_j is temporally consistent if the following two conditions hold: (i) $(t + ts_j) \leq avi_j$ (absolute consistency) and (ii) (assume R is the set for o_j) for all $o_k \in R | \mathrm{ts}_j - \mathrm{ts}_k | \leq R_{rvi}$.

Concurrency Control

A conventional non-real-time two-phase-locking scheme (2PL) is prone to priority inversion, unknown blocking times, and deadlocks, making it infeasible in a real-time context. Priority inversion occurs when a high-priority transaction is blocked due to a resource locked by a low-prioritized transaction. Thus, the high-priority transaction experiences blocking delays that can jeopardize this timeliness. Several 2PL variants have been developed to overcome these deficiencies, and the most well-known scheme is two-phase-locking

with high priority resolution (2PL-HP). Consider the case a transaction T_R is requesting a lock, which is held by another transaction T_H. In 2PL-HP, T_R will abort T_H if the priority of T_R exceeds that of T_H; otherwise T_R will wait until T_H completes. There are additional schemes that deploy more elaborate conditions to avoid that a process is aborted unnecessarily, e.g., considering the remaining execution time of T_R and the needed execution time of T_R.

In real-time optimistic concurrency control (OCC), transactions execute in three stages: read, validation, and write. In the read stage, transactions read and update data items freely, storing their updates into private workspaces. Note that the updates of a transaction stored in the private workspaces are installed as global copies in the write stage (i.e., after the transaction is validated). In the validation stage, a validating transaction may conflict with ongoing transactions. Depending on the transaction characteristics, there are several real-time conflict resolution mechanisms for the validation [5], i.e., a validating transaction may commit, abort, or be "put on the shelf" to wait for the conflicting transactions.

In priority-wait conflict resolution mechanism [5], which is demonstrated to have superior performance for OCC, a transaction that reaches validation and finds higher priority transactions in its conflict set is "put on the shelf", that is, it is made to wait and not allowed to commit immediately. This gives the higher priority transactions a chance to make their deadlines first. After all conflicting higher priority transactions leave the conflict set, either due to committing or due to aborting, the on-the-shelf waiter is allowed to commit. Note that a waiting transaction might be restarted due to the commit of one of the conflicting higher priority transactions.

Distributed Real-Time Transaction Processing The distributed transaction execution model for a real-time two-phase commit protocol is presented next. A commonly adopted sub-transaction model has been presented [4] in which there is one process, called the master, which is executed at the site where the transaction is submitted, and a set of other processes, called cohorts, which execute on behalf of the transaction at the various sites that are accessed by the transaction. Cohorts are created by the master sending a STARTWORK message to the local transaction manager at that site. This message includes the work to be done at that site and is passed on to the cohort. Each cohort sends a WORKDONE message to the master after it has completed its assigned data processing work. The master initiates the commit protocol (only) after it has received this message from all its cohorts. Within the above framework, a transaction may execute in either sequential or parallel fashion. The distinction is that cohorts in a sequential transaction execute one after another, whereas cohorts in a parallel transaction execute concurrently.

Real-Time Two-Phase Commit The master implements the classical two-phase commit protocol [3] to maintain transaction atomicity. In this protocol, the master, after receiving the WORKDONE message from all its cohorts, initiates the first phase of the commit protocol by sending PREPARE (to commit) messages in parallel to all its cohorts. Each cohort that is ready to commit first force-writes a prepare log record to its local stable storage and then sends a YES vote to the master. At this stage, the cohort has entered a prepared state wherein it cannot unilaterally commit or abort the transaction, but has to wait for the final decision from the master. On the other hand, each cohort that decides to abort force writes an abort log record and sends a NO vote to the master. Since a NO vote acts like a veto, the cohort is permitted to unilaterally abort the transaction without waiting for the decision from the master.

After the master receives votes from all its cohorts, the second phase of the protocol is initiated. If all the votes are YES, the master moves to a committing state by force-writing a commit log record and sending COMMIT messages to all its cohorts. Each cohort, upon receiving the COMMIT message, moves to the committing state, force-writes a COMMIT log record, and sends an ACK message to the master. On the other hand, if the master receives one or more NO votes, it moves to the aborting state by force-writing an abort log record and sends ABORT messages to those cohorts that are in the prepared state. These cohorts, after receiving the ABORT message, move to the aborting state, force-write an abort log record and send an ACK message to the master. Finally, the master, after receiving ACKs from all the prepared cohorts, writes an end log record and then "forgets" the transaction (by removing from virtual memory all information associated with the transaction).

A real-time two-phase commit protocol such as PROMPT (Permits Reading Of Modified Prepared

data for Timeliness) [6], works differently from traditional two-phase commit protocol in that transactions requesting data items held by other transactions in the prepared state are allowed to access this data. That is, prepared cohorts lend their uncommitted data to concurrently executing transactions (without releasing the update locks) in the optimistic belief that this data will be committed. If the lender is aborted later, the borrower is also aborted since it has utilized dirty data. On the other hand, if the borrowing cohort completes its local data processing before the lending cohort has received its global decision, the borrower is "put on the shelf", that is, it is made to wait until either the lender receives its global decision or its own deadline expires, whichever happens earlier. In this case, the borrower can only commit if the lender commits.

In contrast to centralized databases where transactions that validate successfully *always* commit, a distributed transaction that is successfully locally validated might be aborted later because it fails during global validation. This can lead to "wasteful" aborts of transactions–a transaction that is locally validated may abort other transactions in this process. If this transaction is itself later aborted during global validation, it means that all the aborts it caused during local validation were unnecessary.

Key Applications

The number of real-time applications that handle real-time data is large. Thus, a range of applications that manage data with temporal constraints is given in the following.

- *Air-traffic control systems.* This system is used to monitor air-traffic around an airport, which requires continuous monitoring of aircraft positions and weather data, as well as (non-real-time) data of different aircraft types.
- *Control system.* A specific case is an engine control system, which approximately uses readings of twenty sensors, and these sensor values are fundamental in the computation of 400 other variables, where the validity of the derived data is a function of the sensed value. Derived data is used control the mixture of fuel and air in the ignition, as well as diagnostics.
- Wireless networking systems for retrieval of subscriber information such as service subscription and device location stored in Home Location Register (HLR) as well as billing information in the case of prepaid phone calls. For example, the network communication protocols have various timers for call delivery that may cause connection failure if information is not retrieved before a transaction deadline.

Future Directions

The increasing number of applications that handle large amounts of real-time data calls for a strong support from the underlying transaction processing system to satisfy timeliness and consistency requirements. The development of time-cognizant concurrency control and scheduling algorithms has provided a foundation for real-time transaction processing systems. There are several challenging areas where progress would be conducive to enforce the predictability, and below a few examples are provided

- The underlying assumption which most real-time scheduling approaches build upon is the knowledge of the worst-case execution times of the transactions, either a priori to the system starts its execution or they are made available upon their arrival to the system. The dynamic nature of transaction workloads can cause the system to experience transient overloads. Techniques that are shown to effectively manage the uncertainty of execution times or can measure tight and not overly conservative worst case execution times would increase schedulability and resource utilization, as well as techniques for resolving and minimizing the effects of transient overloads.
- *Extended transaction models supporting relaxed ACID properties.* Enforcement of ACID in real-time applications has shown to be costly, jeopardize timeliness, and does not utilize the potential parallelism that can be exploited by relaxation of the ACID properties. This has resulted in the development of, e.g., epsilon-serializability and the notion of data similarity. There is a need for additional application-centric consistency that would capture the tolerance of the applications and the presence of multiple models coexisting in parallel.
- *Transaction scheduling for guaranteeing data temporal consistency.* There is need to maintain coherency between the state of the environment and data used in applications such as control systems.

For example, in order to react to abnormal situations in time, it is necessary to monitor the environment continuously. Such requirements pose a great challenge for maintaining the freshness of data while scheduling transactions to meet their deadlines.

- The notion of data precision and data confidence are becoming increasingly important in real-time applications, i.e., data is annotated with additional attributes to represent the confidence in the data value, (i.e., the level of trust in how the data was derived) and the precision/accuracy of the data. It is unclear how these two notions can be effectively used as decision parameters for concurrency control and schedulability.

Cross-references

- ▶ ACID Properties
- ▶ Temporal Consistency

Recommended Reading

1. Abbott R. and Garcia-Molina H. Scheduling real-time transactions: a performance evaluation. In Proc. 14th Int. Conf. on Very Large Data Bases, 1988.
2. Bestavros A. and Fay-Wolfe V. Real-Time Database and Information Systems – Research Advances. Kluwer Academic Publishers, MA, USA, 1997.
3. Carey M. and Livny M. Conflict detection tradeoffs for replicated data. ACM Trans. Database Syst., 16:703–746, 1991.
4. Gray J. and Reuter A. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, CA, USA, 1992.
5. Haritsa J., Carey M., and Livny M. Data access scheduling in firm real-time database systems. J. Real-Time Syst., 4: 203–241, 1992.
6. Haritsa J., Ramamritham K., and Gupta R. The PROMPT real-time commit protocol. IEEE Trans. Parall. Distr. Syst., 11(2):160–181, 2000.
7. Ramamritham K. Real-time databases. Int. J. Distr. Parall. Databases, 1, (2) 1993.
8. Ramamritham K., Son S.H., and Cingiser D.L. Real-Time databases and data services. Real-Time Syst. J., 28:179–215, 2004.
9. Son S.H. Advances in Real-Time Systems. Prentice-Hall, NJ, USA, 1995.
10. Soparkar N., Korth H.F., and Silberschatz A. Time-Constrained Transaction Management – Real-Time Constraints in Database Transaction Systems. Kluwer Academic Publishers, MA, USA, 1996.
11. Stankovic J. and Ramamritham K. What is Predictability for Real-Time Systems? Real-Time Syst., 2(4):247–254, 1990.
12. Xiong M., Han S., and Lam K.-Y. A Deferrable Scheduling Algorithm for Real-Time Transactions Maintaining Data Freshness. In Proc. 26th IEEE Real-Time Systems Symp., 2005, pp. 27–37.

Real-World Time

▶ Valid Time

Reasoning with Qualitative Temporal Constraints

▶ Qualitative Temporal Reasoning

Recall

Ethan Zhang[1,2], Yi Zhang[1]
[1]University of California, Santa Cruz, Santa Cruz, CA, USA
[2]Yahoo! Inc., Santa Clara, CA, USA

Definition

Recall measures the coverage of the relevant documents of an information retrieval (IR) system. It is the fraction of all relevant documents that are retrieved. 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 $|R|$ and $|A|$ be the numbers of documents in R and A, respectively. Let $|R \cap A|$ denote the number of documents that are in both R and A. The *recall* of the IR system for Q is defined as $R = |R \cap A| / |R|$.

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

- ▶ Eleven Point Precision-recall Curve
- ▶ F-Measure
- ▶ Precision
- ▶ Standard Effectiveness Measures

Receiver Operating Characteristic

Pang-Ning Tan
Michigan State University, East Lansing, MI, USA

Synonyms

Operating characteristic; Relative operating characteristic; ROC

Definition

Receiver operating characteristic (ROC) analysis is a graphical approach for analyzing the performance of a classifier. It uses a pair of statistics – true positive rate and false positive rate – to characterize a classifier's performance. The statistics are plotted on a two-dimensional graph, with false positive rate on the x-axis and true positive rate on the y-axis. The resulting plot can be used to compare the relative performance of different classifiers and to determine whether a classifier performs better than random guessing.

Historical Background

ROC analysis was originally developed in signal detection theory to deal with the problem of discriminating known signals from a random noise background [11]. It was first applied to the radar detection problem to quantify how effective targets such as enemy aircrafts can be identified according to their radar signatures. In the 1960s, ROC analysis was applied to experimental psychology and psychophysics [6]. The approach has subsequently found its application in a variety of areas including radiology, epidemiology, finance, weather forecasting, and social sciences. In machine learning, the benefits of using ROC analysis for evaluating and comparing the performance of classifiers was first demonstrated by Spackman [14]. The approach was brought to the attention of the data mining community by Provost and Fawcett [12], who proposed the idea of using a convex hull of ROC curves to compare multiple classifiers in imprecise and changing environments.

Foundations

Predictive accuracy has traditionally been used as the primary evaluation measure for classifiers. However, its limitation is well-documented, particularly for data sets with skewed class distributions [12]. ROC analysis provides an alternative way for measuring performance by examining the trade-off between the successful detection of positive examples and the misclassification of negative examples. The approach was originally developed for binary classification problems, where each example is assigned to either a positive or a negative class. When applying the classifier to a given example, four possible outcomes may arise: (i) true positive (TP), when a positive example is classified correctly, (ii) true negative (TN), when a negative example is classified correctly, (iii) false positive (FP), when a negative example is misclassified as positive, and (iv) false negative (FN), when a positive example is misclassified as negative. These outcomes can be tabulated in a 2×2 table known as the confusion matrix, as shown in Table 1.

An ROC graph is constructed by examining the true positive rate and false positive rate of a classifier. The true positive rate (TPR), also known as hit rate or sensitivity, corresponds to the proportion of positive examples that are correctly labeled by the classifier, whereas the false positive rate (FPR), also known as false alarm rate, corresponds to the proportion of negative examples that are incorrectly labeled. Mathematically, these statistics are computed from a given confusion matrix as follows:

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{1}$$

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{2}$$

The number of points in an ROC graph depends on the type of output produced by the classifier. A classifier that produces a discrete-valued output is mapped to a single point in the ROC graph because there is only one confusion matrix. Other classifiers such as naïve Bayes and neural networks can produce numeric-valued outputs to indicate the degree to which an example belongs to the positive class. A threshold must be specified to determine the class membership – if the classifier's output exceeds the threshold, the example is assigned to the positive class. Each threshold setting leads to a different point in the ROC graph. By varying the threshold, a piecewise linear curve, known as the ROC curve, is formed. For example, the solid line in Fig. 1c shows the ROC curve obtained by varying the threshold on classification outputs produced by a neural network classifier.

There are several critical points in the diagram that are of practical significance. The critical point (FPR = 0,

TPR = 0) corresponds to an extremely conservative classifier, i.e., one that assigns every example to the negative class. In contrast, the critical point (FPR = 1, TPR = 1) corresponds to a classifier that liberally declares every example to be positive. Since the ideal classifier corresponds to the critical point (FPR = 0, TPR = 1), points closer to the upper-left corner of the ROC graph are generally better classifiers. A random classifier, on the other hand, produces points that reside along the diagonal line connecting the bottom-left to the upper-right corner of the ROC graph, as shown by the dashed line in Fig. 1c. For example, a classifier that randomly assigns one-fourth of the examples to the positive class has TPR = 0.25 and FPR = 0.25.

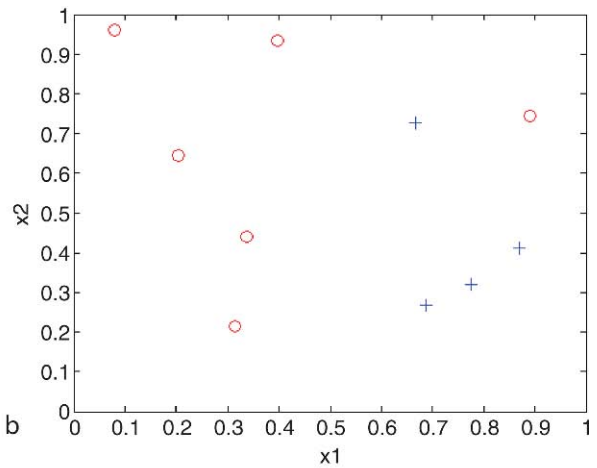
An ROC curve X dominates another ROC curve Y if X lies above and to the left of Y. The more dominant the ROC curve, the better the classifier is. For example, Fig. 2 shows that classifier A is better than classifiers B and C because it has a more dominant ROC curve. In practice, however, one seldom finds an ROC curve that completely dominates other ROC curves. Instead, one would find different ranges of FPR values in which one classifier is better than another. For example,

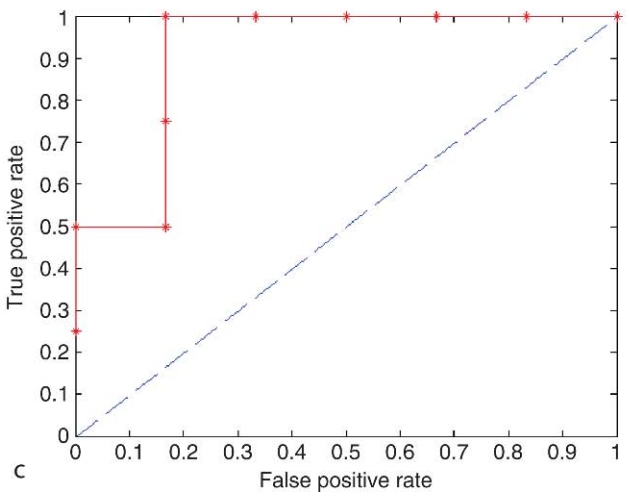
Receiver Operating Characteristic. Table 1. Confusion matrix for a 2-class problem

		Predicted Class	
		+	–
Actual Class	+	TP	FN
	–	FP	TN

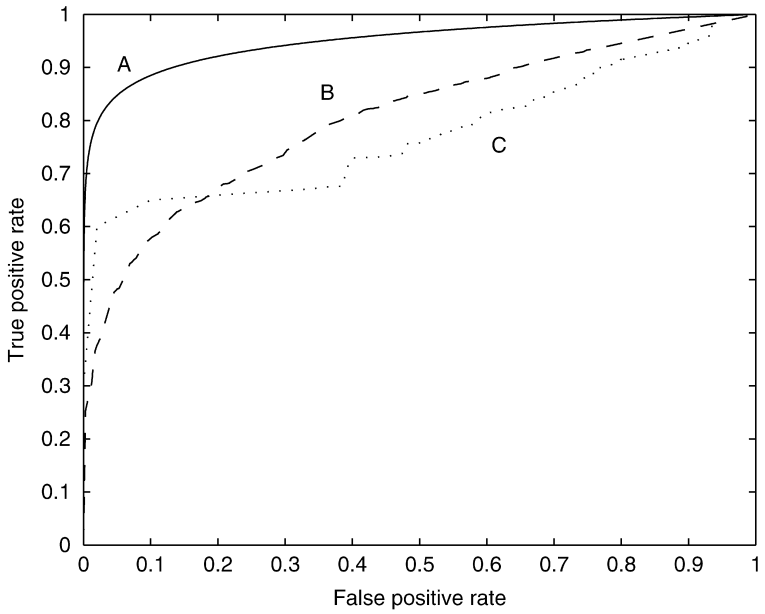
a

x1	x2	Class	Output
0.08	0.96	-	–1.50
0.21	0.64	-	–0.95
0.32	0.21	-	–0.33
0.34	0.44	-	–0.48
0.40	0.93	-	–0.79
0.67	0.73	+	–0.03
0.69	0.27	+	0.42
0.77	0.32	+	0.56
0.87	0.41	+	0.68
0.89	0.74	-	0.43





Receiver Operating Characteristic. Figure 1. ROC curve for a two-dimensional data set classified using a single layer neural network.



Receiver Operating Characteristic. Figure 2. Performance comparison of classifiers using ROC curve.

in Fig. 2, classifier C is better than B when FPR below 0.2 or above 0.95. Therefore, Provost and Fawcett [12] introduced the ROC convex hull (ROCCH) method to combine the ROC curves from a set of classifiers to obtain the most dominant ROC curve. The convex hull can be used to identify ranges of FPR values in which a classifier is potentially optimal.

Although an ROC curve provides a visual display of a classifier's performance, it is often useful to summarize the curve into a single metric to estimate the overall classifier's performance. By analyzing the statistical properties of the metric [10], this provides a quantitative way to determine whether the observed difference in performance of two classifiers is statistically significant. Examples of ROC-derived metrics include area under ROC curve (AUC), slope intercept index, and the ROC break-even point. AUC is also equivalent to the Wilcoxon-Mann-Whitney statistic [7], thus allowing us to compute its statistical properties such as standard error and confidence interval.

In addition to ROC curves, there are alternative ways to visualize the performance of a classifier such as precision-recall curves [2] and cost curves [3]. A precision-recall curve plots the tradeoff between precision, which is the fraction of examples classified as positive that are actually positive, against recall, which is equivalent to true positive rate. Davis and Goadrich [2] has shown that a curve that dominates in the ROC space also dominates in the precision-recall space. Nevertheless, an ROC curve may not effectively capture important differences between classifiers when applied to data sets with skewed class distributions. For example, the confusion matrices shown in Table 2 have very similar TPR and FPR values even though their precision values are very different. The ROC representation also does not commit to any particular cost function or class distribution. As a result, it does not convey information such as the misclassification costs and class probabilities for which a classifier performs better than another. To overcome this limitation, Drummond and Holte [3] proposed the idea of using cost curves to explicitly represent the cost information.

Receiver Operating Characteristic. Table 2. Confusion matrices to illustrate the difference between ROC and precision-recall curves

(a)		Predicted Class	
		+	–
Actual Class	+	35	5
	–	25	435
(b)		Predicted Class	
		+	–
Actual Class	+	35	5
	–	5	455

A cost curve plots the expected cost of a classifier against a probability cost function which is defined as follows:

$$\text{Probability Cost Function} = \frac{P(+)C(-|+)}{P(+)C(-|+) + P(-)C(+|-)} \quad (3)$$

where $P(+)$ and $P(-)$ are the prior probabilities of each class, $C(-|+)$ is the cost of misclassifying a positive example as negative, while $C(+|-)$ is the cost of misclassifying a negative example as positive.

Key Applications

ROC analysis has been successfully applied to many application domains including psychology (in the studies of perception to resolve the issue of sensory threshold) [15], radiology (to distinguish between subjective judgment and objective detectability in imaging systems) [9], and epidemiology [13]. In addition to classification problems, ROC analysis is also applicable to other ranking problems such as recommender systems, information retrieval, and anomaly detection.

Future Directions

Most of the previous work on ROC analysis are limited to binary class problems. For multi-class problems, the analysis is more complicated as the confusion matrix is no longer a simple 2×2 table. An obvious way to extend the approach is to generate a different ROC graph for each class. However, with this approach, the ROC analysis is no longer insensitive to the class distribution [4]. Furthermore, combining the AUC statistics from multiple ROC graphs remains an open problem.

Another research direction that has attracted considerable interests in recent years is designing classification algorithms that directly optimize the area under ROC curve, or equivalently, the Wilcoxon-Mann-Whitney statistic. For instance, Cortes and Mohri [1] showed that, under certain conditions, the objective function optimized by the RankBoost algorithm is identical to AUC. New algorithms have also been developed to incorporate AUC into decision tree induction [5] and support vector machines [8].

URL to Code

The WEKA data mining software provides codes for plotting ROC curves and cost curves (http://www.cs.waikato.ac.nz/~ml/weka/.) The software for plotting ROC convex hull is available at http://home.comcast.net/~tom.fawcett/public_html/ROCCH/index.html.

Cross-references

▶ Area Under ROC Curve
▶ Classification

Recommended Reading

1. Cortes C. and Mohri M. Auc optimization vs. error rate minimization. In Advances in Neural Inf. Proc. Syst. 16, Proc. Neural Inf. Proc. Syst., December 2003.
2. Davis J. and Goadrich M. The relationship between precision-recall and roc curves. In Proc. 23rd Int. Conf. on Machine Learning, 2006.
3. Drummond C. and Holte R.C. Explicitly representing expected cost: an alternative to roc representation. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 198–207.
4. Fawcett T. An introduction to roc analysis. Pattern Recogn. Lett., 27(8):861–874, 2006.
5. Ferri C., Flach P.A., and Hernandez-Orallo J. Learning decision trees using the area under the roc curve. In Proc. 19th Int. Conf. on Machine Learning, 2002.
6. Green D.M. and Swets J.A. (eds.). Signal Detection Theory and Psychophysics. Wiley, New York, 1966.
7. Hanley J.A. and McNeil B.J. The meaning and use of the area under a receiver operating characteristic (roc) curve. Radiology, 143(1):29–36, 1982.
8. Joachims T. A support vector method for multivariate performance measures. In Proc. 22nd Int. Conf. on Machine Learning, 2005.
9. Lusted L.B. Signal detectability and medical decision making. Science, 171, 1971.
10. McNeil B.J. and Hanley J.A. Statistical approaches to the analysis of the receiver operating characteristic (roc) curves. Med. Decis. Making, 4(2):137–150, 1984.
11. Peterson W.W., Birdsall T.G., and Fox W.C. The theory of signal detectability. IRE Trans., PGIT-4, 1954.
12. Provost F.J. and Fawcett T. Analysis and visualization of classifier performance: comparison under imprecise class and cost distributions. In Proc. 3rd Int. Conf. on Knowledge Discovery and Data Mining, 1997, pp. 43–48.
13. Sackett D.L. Clinical diagnosis and the clinical laboratory. Clin. Invest. Med., 1, 1978.
14. Spackman K.A. Signal detection theory: Valuable tools for evaluating inductive learning. In Proc. 6th Int. Workshop on Machine Learning, 1989, pp. 160–163.
15. Swets J.A. The relative operating characteristics in psychology. Science, 182, 1973.

Recodings

▶ Matrix Masking

Reconciliation-based Data Replication

► Optimistic Replication and Resolution

Record Extraction

► Column Segmentation

Record Linkage

Josep Domingo-Ferrer
Universitat Rovira i Virgili, Tarragona, Catalonia

Synonyms

Record matching; Re-identification

Definition

Record linkage is a computational procedure for linking each record a in file A (e.g., a file masked for disclosure protection) to a record b in file B (original file). The pair (a, b) is a match if b turns out to be the original record corresponding to a.

Key Points

Record linkage techniques were created for data fusion and to increase data quality. However, they have also found an application in measuring the risk of identity disclosure in statistical disclosure control. In the SDC context, it is assumed that an intruder has an external dataset sharing some (key or outcome) attributes with the released protected dataset and containing additionally some identifier attributes (e.g., passport number, full name, etc.). The intruder is assumed to attempt to link the protected dataset with the external dataset using the shared attributes. The number of matches gives an estimation of the number of protected records whose respondent can be re-identified by the intruder. Accordingly, disclosure risk is defined as the proportion of matches among the total number of records in A.

There are two main types of record linkage used to measure identity disclosure in SDC: distance-based record linkage and probabilistic record linkage.

Distance-based record linkage consists of linking each record a in file A to its nearest record b in file B. Therefore, this method requires a definition of a distance function for expressing *nearness* between records. This record-level distance can be constructed from distance functions defined at the level of attributes. Construction of record-level distances requires standardizing attributes to avoid scaling problems and assigning each attribute a weight on the record-level distance. A straightforward choice is to use the Euclidean distance, but other distances can be used.

The main advantages of using distances for record linkage are simplicity for the implementer and intuitiveness for the user. Another strong point is that subjective information (about individuals or attributes) can be included in the re-identification process by properly modifying distances.

The main difficulty of distance-based record linkage consists of coming up with appropriate distances for the attributes under consideration. For one thing, the weight of each attribute must be decided and this decision is often not obvious. Choosing a suitable distance is also especially difficult in the cases of categorical attributes and of masking methods such as local recoding where the masked file contains new labels with respect to the original dataset.

Like distance-based record linkage, probabilistic record linkage aims at linking pairs of records (a, b) in datasets A and B, respectively. For each pair, an index is computed. Then, two thresholds LT and NLT in the index range are used to label the pair as linked, clerical or non-linked pair: if the index is above LT, the pair is linked; if it is below NLT, the pair is non-linked; a clerical pair is one that cannot be automatically classified as linked or non-linked and requires human inspection. When independence between attributes is assumed, the index can be computed from the following conditional probabilities for each attribute: the probability $P(1|M)$ of coincidence between the values of the attribute in two records a and b given that these records are a real match, and the probability $P(0|U)$ of non-coincidence between the values of the attribute given that a and b are a real unmatch.

To use probabilistic record linkage in an effective way, one needs to set the thresholds LT and NLT and estimate the conditional probabilities $P(1|M)$ and $P(0|U)$ used in the computation of the indices. In plain words, thresholds are computed from: (i) the probability $P(LP|U)$ of linking a pair that is an unmatched pair (a *false positive* or *false linkage*) and (ii) the probability $P(NP|M)$ of not linking a pair that is a match (a *false negative* or *false unlinkage*). Conditional

probabilities $P(1|M)$ and $P(0|U)$ are usually estimated using the EM algorithm.

Cross-references

- ▶ Disclosure Risk
- ▶ Inference Control in Statistical Databases
- ▶ Microdata
- ▶ Record Matching

Recommended Reading

1. Fellegi I.P. and Sunter A.B. A theory for record linkage. J. Am. Stat. Assoc., 64(328):1183–1210, 1969.
2. Torra V. and Domingo-Ferrer J. Record linkage methods for multidatabase data mining. In Information Fusion in Data Mining, V. Torra (ed.). Germany, Springer, Berlin. 2003, pp. 101–132.

Record Matching

Arvind Arasu[1], Josep Domingo-Ferrer[2]
[1]Microsoft Research, Redmond, WA, USA
[2]The Public University of Tarragona, Tarragona, Spain

Synonyms

Record linkage; Merge-purge; Entity resolution; Data deduplication; Duplicate detection; Instance identification; Name matching

Definition

Record matching is the problem of identifying whether two records in a database refer to the same real-world entity. For example, in Fig. 1, the customer record $A1$ in Table A and record $B1$ in Table B probably refer to the same customer, and should therefore be matched. (The example in Fig. 1 was adapted from an example in [21].) As Fig. 1 suggests, the same entity can be encoded in different ways in a database; this phenomenon is fairly common and occurs due to a variety of natural reasons such as different formatting conventions, abbreviations, and typographic errors. Record matching is often studied in the following setting: Given two relations A and B, identify all pairs of matching records, one from each relation. For the two tables in Fig. 1, a reasonable output might be the pairs $(A1,B1)$ and $(A2,B2)$. In some settings of the record matching problem, there is a constraint that each record of table A be matched with at most one record of table B. This asymmetric setting is typically used when records in table B are "clean" and those of table A, "dirty." The record matching problem is closely related to the *deduplication* problem. The focus of record matching is to identify pairs of matching records while that of deduplication is to partition records so that records in the same partition refer to the same entity. In practice, the output produced by record matching is inaccurate; it is not an equivalence relation and does not correspond to a natural partitioning. The above distinction between record matching and deduplication is often ignored and the two terms are used synonymously.

Historical Background

Record matching has a rich history dating back to the work by Newcombe and others [16] in 1959. Fellegi and Sunter [10] formalize the intuition of Newcombe and others. Specifically, they cast the record matching problem as a classification problem that can be stated as follows: Given a vector of *similarity scores* between attribute values for a pair of records, classify the pair as a *match* or a *nonmatch*. For a given attribute, a similarity score indicates how similar the two records are on that attribute. A simple similarity measure assigns a score 1 if the records agree on the attribute and 0 otherwise. More sophisticated similarity measures are discussed subsequently. Fellegi and Sunter [10] use a naive Bayes classifier, but subsequent work has considered other kinds of classifiers such as decision trees [7] and SVMs [3]. The classifiers are typically trained using learning examples comprising of a set of record pairs, each labeled as a match or a non-match [3,7,10]. One of the problems with this approach is that learning examples required to train an accurate classifier are hard to generate, since they should not be either obvious matches or obvious non-matches. Sarawagi and Bhamidipaty [17] address this problem and propose an approach based on active learning to interactively identify useful learning examples. Jaro [13] and Winkler [20] propose an alternate approach where the classifier is learned in an unsupervised setting without learning examples using a variant of EM (expectation maximization) algorithm.

A large class of work on record matching has focused on more sophisticated measures of similarity between attribute values of records. As mentioned above, these form the basis for classifying record pairs as a match or a nonmatch. A variety of classic similarity functions such as edit distance and variants [5] and cosine similarity with

a

Id	Name	Address	Age
A1	J H Smith	16 Main St	17
A2	Javeir Marteenez	49 Apple cross Road	33
A3	Gillian Jones	645 Reading Ave	24

b

Id	Name	Address	Age
B1	John H Smith	16 Main Street	17
B2	Javier Martinez	49 E Apple cross Road	33
B3	Jilliam Brown	123 Norcross Blvd	43

Record Matching. Figure 1. Record matching example.

tf-idf (term frequency-inverse document frequency) weights [8] have been used for record matching. Jaro [12] proposes a more domain specific similarity measure designed for people names. Sarawagi and Bhamidipaty [17] propose using a weighted linear combination of simple similarity functions such as those mentioned above, and present techniques for learning the weights using training examples. Bilenko and Mooney [3] present a generalization of edit distance whose parameters can be learnt from a large text corpus. Arasu and others [1] present a framework for *programmable similarity*, where a similarity function can be programmed to be sensitive to synonymous words or phrases such as `Robert` and `Bob` or `US` and `United States`.

Another body of work on record matching has focused on efficiency issues. For large inputs, it is impractical to exhaustively consider all pairs of records and check if they are a match or not. Hernandez and Stolfo [11] present the *sorted neighborhood approach*, which linearly orders all the records based on a carefully selected key and considers only pairs of records that are close to each other in the linear ordering. McCallum and others [15] present an approach based on *canopies*, which are overlapping clusters of the input records. Only pairs of records within a cluster are checked for a match. Chaudhuri and others [6] identify *set-similarity join* as a useful primitive for large scale record matching.

Foundations

String Similarity

Most record matching approaches are based on the observation that two matching records have similar values for their attributes. For example, the records A2 and B2 have similar names and similar addresses and the same age, and are therefore likely to be matches. The similarity between two values is typically determined using a *similarity function* that takes two values and produces as output a number that quantifies the similarity of the values. String similarity functions that quantify the similarity between two strings are particularly relevant for record matching since many attributes in record matching are textual in nature. One of the earliest and well-known string similarity measure is *edit distance* or *Levenshtein distance*. The edit distance between two strings $s1$ and $s2$ is defined as the smallest number of edit operations required to produce $s2$ from $s1$, where an edit operation is an insertion, deletion, or substitution at the character level. For example, the edit distance between `Martinez` and `Marteenez` is 2 since one can derive the second string from the first using one substitution and one insertion. Edit distance is often a poor fit for record matching since two strings representing different entities can have small edit distance (e.g., `148th Ave NE` and `147th Ave NE`) and two strings representing the same entity, a large edit distance (e.g., `148th Ave NE` and `148th Ave Northeast`).

An alternate approach that works well for some domains is to treat strings as a bag of words (or *tokens*) and use a token-based similarity function such as *cosine* or *jaccard* (defined below). For a string s, let $Tokens(s)$ denote the set of tokens in s. For a token t, let $w(t)$ denote the *weight* of token t. The weight of a token represents its "importance" for the purposes of computing similarity. The jaccard similarity of two strings s_1 and s_2 is defined as:

$$\frac{|\ Tokens(s_1) \cap Tokens(s_2)\ |}{|\ Tokens(s_1) \cup Tokens(s_2)\ |}$$

For a set S, $|S|$ denotes the weighted cardinality of S, i.e., the sum of weights of the tokens in B. The cosine similarity between two strings s_1 and s_2 is defined as:

$$\frac{|| \ Tokens(s_1) \cap Tokens(s_2) \ ||_2}{|| \ Tokens(s_1) \ ||_2 \cdot || \ Tokens(s_2) \ ||_2}$$

For a set S, $||S||_2$ is defined as $\sqrt{\sum_{t \in S} w(t)^2}$. Token-based similarity functions differ from edit distance in two aspects: First, they ignore the ordering of tokens; for example, `148th Ave NE` represents the same bag of tokens as `NE 148th Ave`. This feature is desirable in some domains. For example, the list of authors appears before the title string in some citations and after the title in other citations. An order-sensitive similarity function would produce a low similarity score for two citation strings referring to the same publication but with different author-title orderings. Second, token-based similarity functions are not sensitive to intra-token edits. Two tokens are considered different even if they are textually very similar. As mentioned earlier, this feature is useful in domains such as addresses where, for example, two street names can differ by a single character. Token-based similarity functions enable weighting of tokens to capture their relative importance. A commonly used weighting scheme is the idf-based one, where the weight $w(t)$ of a token t is defined to be $log(N/N_w)$, where N is the number of records in a reference table and N_w denotes the number of records in the reference table containing the token t. With idf-based weighting, the similarity of the pair (`Applecross Rd`, `Applecross Road`) would be higher than the similarity of the pair (`Applecross Rd`, `Maltby Rd`) although both pairs have one token that is common to the two strings in the pair. This happens since the idf-weights of the rarer words `Applecross` and `Maltby` are higher than the weights for the common words `Road` and `Rd`. A related class of similarity functions is obtained by tokenizing strings to their character-level n-grams instead of words. For example, the set of 2-grams of the string `Applecross` is: `{Ap, pp, pl, le, ec, cr, ro, os, ss}`. These similarity functions share some characteristics with edit distance since they can capture intra-word edits, but they are not as order sensitive as edit distance.

A variety of other domain specific similarity functions have been used for record matching. These are covered in depth in the survey by Elmagarmid and others [9] and the tutorial by Koudas and others [14].

Record Matching

Let R and S denote the input tables of record matching. The record matching problem can be viewed as a binary classification problem, where a given pair of records (r, s), $r \in R$ and $s \in S$, has to be classified either as a *match* or a *nonmatch*. A slight variant, not discussed here, is to consider a third category *possible match* consisting of hard-to-classify pairs that require manual inspection. One common approach for record matching is to train a standard binary classifier using learning examples consisting of a set of record pairs prelabeled as a match or nonmatch. The original Fellegi and Sunter approach uses a naive Bayes classifier and is discussed below. Other kinds of classifiers such as SVMs [3] and decision trees [7] have also been used.

Fix a record pair $(r, s) \in R \times S$. Define a *similarity vector* $\overline{x} = [x_1, \ldots, x_n]$, where each x_i denotes the similarity between r and s on some attribute A. The similarity can be computed using one of the functions discussed earlier (for string attributes). It can also be a simple binary value based on equality, i.e., $x_i = 1$ if r and s agree on the attribute and 0, otherwise. Let M denote the event that the pair (r,s) is a match and U the event that it is a nonmatch. Using Bayes rule, one gets

$$Pr(M|\overline{x}) = \frac{Pr(\overline{x}|M)Pr(M)}{Pr(\overline{x})}$$

$$Pr(U|\overline{x}) = \frac{Pr(\overline{x}|U)Pr(U)}{Pr(\overline{x})}$$

Therefore, $Pr(M|\overline{x}) \geq Pr(U|\overline{x})$ whenever,

$$\frac{Pr(\overline{x}|M)}{Pr(\overline{x}|U)} \geq \frac{Pr(U)}{Pr(M)}$$

The expression $\ell(\overline{x}) = Pr(\overline{x}|M)/Pr(\overline{x}|U)$ is called the *likelihood function*. The construction of the naive Bayes classifier assumes that x_i and x_j, $i \neq j$, are conditionally independent given M or U, implying:

$$Pr(\overline{x}|M) = \prod_{i=1}^{n} p(x_i|M)$$

$$Pr(\overline{x}|U) = \prod_{i=1}^{n} p(x_i|U)$$

The probabilities $p(x_i|M)$ and $p(x_i|U)$ can be estimated using the learning examples, which can be used to estimate $Pr(\overline{x}|M)$ and $Pr(\overline{x}|U)$ using the expressions above. It is harder to estimate $Pr(U)$ and $Pr(M)$. A simple approach is to empirically pick a threshold

T and classify the pair (r, s) as a match if $\ell(\overline{x}) \geq T$ and a nonmatch otherwise.

A simple classifier that has performance advantages over more sophisticated machine-learning (ML) classifiers is the *threshold-based* classifier. Here, the record pair (r, s) is classified as a match if the similarity vector $\overline{x}$ satisfies:

$$(x_1 \geq T_1) \wedge (x_2 \geq T_2) \wedge \cdots \wedge (x_n \geq T_n)$$

where T_1,...,T_n are thresholds that can be learned using labeled examples or set manually based on domain expertise. Record matching based on a threshold-based classifier can be performed efficiently using the string similarity join primitive discussed below. Chaudhuri and others [4] show that the record matching accuracy of union of several threshold-based classifiers is comparable to that of state-of-art ML classifiers.

A related approach, sometimes called *distance-based* [19] record matching, is to define a distance measure between two records by suitably combining the similarity (distance) scores of the attributes of the two records. A pair (r, s) is considered a match if the distance between r and s is smaller than a given threshold value. This approach is also easily extended to the asymmetric version of record matching where each record in R is constrained to match at most one record in S: For each record r in R, the record in S with the smallest distance to r is selected as a match.

More details of the other approaches used for record matching can be found in the survey by Elmagarmid et al. [9].

Performance

For large input tables R and S, it is impractical to consider every pair of records $(r, s) \in R \times S$ and classify them as a match or a nonmatch. Therefore, an important challenge in record matching is to identify the matching pairs without exhaustively considering every pair in $R \times S$. An important primitive for efficient record matching is *string similarity join*: Given two tables R and S, identify all pairs of records $(r, s) \in R \times S$ such that the string similarity of $r.A$ and $s.B$ is above some specified threshold, where the string similarity is measured using one of the similarity functions mentioned earlier. A string similarity join can be used to heuristically identify promising matching candidates, which can then be subjected to more complex classification. The string similarity join primitive can also be used to perform record matching based on the threshold-based classifier discussed above [4].

Since many of the string similarity measures such as jaccard and cosine are set-based, a more foundational primitive is *set-similarity join* [6]. A set-similarity join conceptually takes two collections of sets U and V as input and outputs all pairs of sets $(u, v) \in U \times V$ having high set-similarity. String-similarity joins can be evaluated using set-similarity joins as a primitive even for many non-set based similarity functions such as edit distance. A simple algorithm for set-similarity join involves building an inverted index over the collection V. The inverted index efficiently retrieves for any given element e all sets in V containing e. For each set u in U, the inverted index is used to retrieve all sets v in V that share one or more elements with u; the pair (u, v) is then produced as output if its satisfies the set-similarity join condition. More efficient algorithms for set-similarity joins are presented in [2,18].

Key Applications

The primary application of record matching is data cleaning. The presence of duplicate records in a database adversely affects the quality of the database and its utility for analysis and mining. Another related application of record matching is data integration. The same real-world entity is often represented differently in other databases being integrated, and record matching is necessary to evolve a common representation for the entity. Another application of record matching is to measure the risk of identity disclosure in statistical disclosure control (SDC) [19]. In the SDC context, a dataset A (for public release) is produced from a source dataset B by masking values for identity protection. The goal of this step is to make it hard for an adversary to link a record in A to the record in B from which it was generated. The number of records in A that can be matched to records in B using the record matching techniques discussed above gives an estimate for disclosure risk of the released dataset A. Distance-based record matching is often used for this purpose.

Cross-references

- ▶ Column segmentation
- ▶ Constraint-Driven Database Repair
- ▶ Data Cleaning
- ▶ Data Deduplication
- ▶ Deduplication in Data Cleaning
- ▶ Record Linkage

Recommended Reading

1. Arasu A., Chaudhuri S., and Kaushik R. Transformation-based framework for record matching. In Proc. 24th Int. Conf. on Data Engineering, 2008, pp. 40–49.
2. Arasu A., Ganti V., and Kaushik R. Efficient exact set-similarity joins. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 918–929.
3. Bilenko M. and Mooney R.J. Adaptive duplicate detection using learnable string similarity measures. In Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003, pp. 39–48.
4. Chaudhuri S., Chen B.C., Ganti V., and Kaushik R. Example-driven design of efficient record matching queries. In Proc. 33rd Int. Conf. on Very Large Data Bases, 2007, pp. 327–338.
5. Chaudhuri S., Ganjam K., Ganti V., and Motwani R. Robust and efficient fuzzy match for online data cleaning. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2003, pp. 313–324.
6. Chaudhuri S., Ganti V., and Kaushik R. A primitive operator for similarity joins in data cleaning. In Proc. 22nd Int. Conf. on Data Engineering, 2006.
7. Cochinwala M., Kurien V., Lalk G., and Shasha D. Efficient data reconciliation. Inf. Sci., 137(1–4):1–15, 2001.
8. Cohen W.W. Data integration using similarity joins and a word-based information representation language. ACM Trans. Inform. Syst., 18(3):288–321, 2000.
9. Elmagarmid A.K., Ipeirotis P.G., and Verykios V.S. Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng., 19 (1):1–16, 2007.
10. Felligi I.P. and Sunter A.B. A theory for record linkage. J. Am. Stat. Soc., 64(328):1183–1210, 1969.
11. Hernandez M. and Stolfo S. The merge/purge problem for large databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1995, pp. 127–138.
12. Jaro M.A. Unimatch: A Record Linkage System: User's Manual. Tech. rep., US Bureau of the Census, Washington DC, 1976.
13. Jaro M.A. Advances in record-linkage methodology as applied to matching the 1985 census of Tampa, Florida. J. Am. Stat. Assoc., 84(406):414–420, 1989.
14. Koudas N., Sarawagi S., and Srivastava D. Record linkage: similarity measures and algorithms. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 802–803.
15. McCallum A., Nigam K., and Ungar L.H. Efficient clustering of high-dimensional data sets with application to reference matching. In Proc. 6th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2000, pp. 169–178.
16. Newcombe H.B., Kennedy J.M., Axford S.J., and James A.P. Automatic linkage of vital records. Science, 130:954–959, 1959.
17. Sarawagi S. and Bhamidipaty A. Interactive deduplication using active learning. In Proc. 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2002, pp. 269–278.
18. Sarawagi S. and Kirpal A. Efficient set joins on similarity predicates. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 743–754.
19. Torra V. and Domingo-Ferrer J. Record Linkage methods for multidatabase data mining. In Information Fusion in Data Mining, V. Torra (ed.), Springer, 2003, pp. 101–132.
20. Winkler W. Improved Decision Rules in the Felligi-Sunter Model of Record Linkage. Tech. rep., Statistical Research Division, US Bureau of the Census, Washington DC, 1993.
21. Winkler W. The state of record linkage and current research problems. Tech. rep., Statistical Research Division, US Bureau of the Census, Washington DC, 1999.

Records Management

▶ Enterprise Content Management

Recovery Guarantees

▶ Application Recovery

Recovery in Distributed Commit Protocols

▶ Distributed Recovery

Recovery in Distributed Database Systems

▶ Distributed Recovery

Recovery in Replicated Database Systems

▶ Distributed Recovery

Recovery Manager

▶ Logging/Recovery Subsystem

Recursive Query Evaluation

▶ Query Processing in Deductive Databases

Recursive View Maintenance

► Maintenance of Recursive Views

Redo

► Crash Recovery
► Logging and Recovery

Redundant Arrays of Independent Disks

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Array; Disk array; Storage array; RAID

Definition

A set of disks from one or more commonly accessible disk subsystems is combined with a body of control software in which part of the physical storage capacity is used to store redundant information about user data stored on the remainder of the storage capacity. The redundant information enables regeneration of user data in a storage emergency in which a disk in the array or an access path fails.

Key Points

The term *RAID* was adopted from the 1988 SIGMOD paper "A Case for Redundant Arrays of Inexpensive Disks (RAID)." In the paper, RAID refers to a group of storage schemes that divide and replicate data among multiple disks to provide better performance, cost and power consumption rate with reasonable availability comparing with conventional Single Large Expensive Disk (SLED) used for Mainframe. Currently, the term of "independent" is usually used rather than "inexpensive" because SLED becomes obsolete.

A number of standard schemes have evolved which are referred to as *levels*. Originally, five RAID levels were conceived, but many more variations have evolved. Currently, there are several sublevels as well as many non-standard levels.

The two key concepts in RAID are striping, and error correction using parity. The striping of RAID means dividing user data into fixed length blocks (striping units). The error correction using parity conceals one disk failure in an array of disks from the host computer and user data lost by one disk failure is regenerated using the parity.

Original five RAID levels are as follows:

1. Level 1: Mirrored disk. Two or more identical copies of data are maintained on separate disks.
2. Level 2: Using Hamming Code and striping with small striping unit (each read or write spread across all disks in an array). During read or write, on the fly ECC is adopted using Hamming Code in order to conceal some number of disk failures in an array.
3. Level 3: Using parity and striping with small striping unit (each read or write spread across all disks in an array). If one disk fails in an array, data on the failed disk are regenerated using corresponding parity and data on other disks in the array.
4. Level 4: Using parity and striping with large striping unit (small read or write can fit in one disk). One disk in an array is allocated to store the parity and user data is striped across remaining disks.
5. Level 5: Using parity and striping with large striping unit (small read or write can fit in one disk). Parity and user data is striped across all disks in an array.

In addition to these 5 levels, Level 0 is often used to represent arrays using striping without data redundancy in which fixed-length sequences of virtual disk data addresses are mapped to sequences of member disk addresses in a regular rotating pattern.

Cross-references

► Disk

Recommended Reading

1. Patterson D., Gibson G., and Katz R. A case for redundant arrays of inexpensive disks (RAID). In Proc. ACM SIGMOD Conf. on Management of Data, 1988.

Reference

► Citation

Reference Collections

► Archiving Experimental Data

Reference Knowledge

Chintan Patel, Chunhua Weng
Columbia University, New York, NY, USA

Definition

Reference knowledge is the knowledge about a particular part of the world in a way that is independent from specific objectives, through a theory of the domain [2]. Different knowledge bases reuse or extend subsets of the reference knowledge for application specific tasks.

Key Points

Developing knowledge bases is a laborious process involving domain experts, knowledge engineers and computer scientists. To minimize this effort, often the core theory and concepts of a given domain are represented in a reference knowledgebase that are reused or extended by other application specific knowledge bases. Consider for example, in the biomedical domain, the Foundational Model of Anatomy [3] is considered a reference knowledge base for representing anatomy terms that can be used for representing the mouse anatomy.

One of the important advantages of using reference knowledge bases is that they provide interoperability [1] across the applications using the knowledge. The applications that heavily reuse or extend the reference knowledge achieve higher levels of interoperability.

The reference knowledge bases are generally developed without any application requirements or objectives. The reference knowledge imposes a rigid set of constraints on representing the application specific or local knowledge base. Such constraints sometimes conflict with the application or turn out to be difficult to implement or maintain, hence impeding the use of reference knowledge. Secondly, the reference knowledge bases tend to be very large [1] and creating subsets appropriate for small applications is challenging.

Recommended Reading

1. Brinkley J., Suciu D., Detwiler L., Gennari J., and Rosse C. A framework for using reference ontologies as a foundation for the semantic web. In Proc. AMIA Annual Symposium, 2006.
2. Burgun A. Desiderata for domain reference ontologies in biomedicine. J Biomed Inform, 39(3):307–313, 2003.
3. Rosse C., and Mejino J. A reference ontology for biomedical informatics: the foundational model of anatomy. J Biomed Inform. 36(6):478–500, 2003.

Reference Reconciliation

► Deduplication in Data Cleaning

Refinement

► Specialization and Generalization

Region Algebra

Matthew Young-Lai
Sybase iAnywhere, Waterloo, ON, Canada

Definition

A region algebra is a collection of operators, each of which returns a set of regions as a result and takes as arguments one or more sets of regions. A *region* of a string is a pair of natural number positions (s,e) that correspond to the substring starting at s and ending at e. A position is a count of bytes, characters, or words from the beginning of the string.

Choosing a set of operators defines a particular region algebra. Operators are chosen for efficiency as well as utility. For example, if regions correspond to structure elements such as chapters and sections in a document, then many operators for querying structure conditions are useful and can be implemented efficiently. One example of such an operator is $\mathit{containedIn}(X,Y)$ which takes two sets of regions X and Y and returns the subset of regions in X that are contained in some region of Y, i.e., $\{(s_x, e_x) \in X \mid \exists (s_y, e_y) \in Y\,(s_x \geq s_y) \wedge (e_x \leq e_y)\}$. A similar operator is $\mathit{contains}(X,Y)$ which returns the subset of regions in X that contain some region of Y, i.e., $\{(s_x, e_x) \in X \mid \exists (s_y, e_y) \in Y\,(s_x \leq s_y) \wedge (e_x \geq e_y)\}$.

Another defining characteristic of a particular region algebra is the restrictions that apply to the sets of regions.

If sets are unrestricted, then a string of length n has $\binom{n}{2}$ regions. This means that the worst case cost of evaluating a single operator cannot be linear in the length of the string. Thus, region algebras are often defined with restrictions on the nesting or overlap of regions in a set.

Historical Background

The first use of a region algebra for text search was in the PAT system [10]. The operators in PAT assume the use of a PAT array (also known as a suffix array) as the underlying implementation. This data structure can provide sets of matches for various lexical patterns (e.g., find all positions corresponding to a given substring of the text). It can also filter matches based on frequency or length conditions (e.g., return only the most frequent words matching a substring). The query language includes operators for dynamically combining matches into regions – a very flexible capability. It also provides set operators such as *union* and *difference* and structure operators such as *including*. Thus, it is a good example of the idea of using a region algebra to mix structure and content operations in text search. The system makes a distinction between sets of regions and sets of points which means that it does not completely fit the definition of a region algebra. Its operators are typed in that their arguments and return values must be either sets of regions or sets of points. This means that the language is not fully compositional. Also, region sets are not allowed to include nesting or overlapping regions which causes semantic problems. For example, when an operator such as *union* results in overlapping regions, only the start points are returned.

Burkowski describes a region algebra for combined content and structure search in text [2,1]. It treats both single words and structure elements spanning many words uniformly as regions. Thus it avoids the problems that result from distinguishing points and regions. Like PAT, region sets do not include nesting or overlapping regions. Later extensions to this work allow regions to overlap which is just as efficient [3]. An advantage of overlapping regions is that it makes the ability to dynamically define regions outside of a fixed hierarchical structure even more flexible and useful. The earlier papers by Burkowski describe simple structure operations such as *containing*, simple content operators for selecting words, and ranking operators based on inverse document frequency from traditional information retrieval. Later work explores ranking for information retrieval in more depth [6].

Jaakkola continues the pattern of imposing fewer restrictions on region sets [7,8]. Sets are allowed to nest and overlap arbitrarily, abandoning the guarantee of linear size relative to the length of the string. However, the maximum depth of nesting tends to be constant in most data, and independent of the length of the data. This is true, at least, when talking about structure in text documents where region algebras are usually applied. Therefore, allowing arbitrary sets gives even more flexibility while not abandoning efficiency in practice.

Compared to relational algebra, there has been relatively little work exploring expressiveness or optimization issues with region algebras. Many systems make an informal effort to balance expressiveness with efficiency. For many useful algebras, the operators are restricted enough that the efficiency of evaluating arbitrary compositions is obvious without need of formal proof. Some work does explore more general issues. For example, Consens and Milo examine equivalence testing and whether operators like direct inclusion can be efficiently supported [4,5]. Young-Lai and Tompa look at characterizing operators that allow the possibility of efficient evaluation [12].

Foundations

The operators of a region algebra can combine and select regions. An important question, however, is where the regions originate. The most flexible approach is to scan the string for matches at every query. These matches can then be treated as regions and further manipulated using the algebra. In principle, any type of pattern language can be used for scanning. For example, it is possible to search for simple substrings, for regular expressions, or for words (possibly with linguistic processing such as stemming or lemmatization). It is also possible to parse the string with a grammar and return structures as regions. This is useful for data such as programming language source code where a well-defined grammar exists. For a scanning operation such as parsing with a grammar, the functionality of the parser starts to overlap with what can be accomplished with the region algebra. In fact, it is possible to use appropriately defined region algebra operators to simulate the process of parsing a string of tokens with a grammar.

If a string is too long to efficiently scan at every query, then regions can be pre-computed and stored persistently. This means making choices before-hand about what queries to support, and involves a tradeoff

between flexibility and space. For example, consider storing a list of regions for every unique word. It is possible to apply a stoplist of common words to save space, but then it is impossible to search for phrases involving these stop words. There is also a tradeoff between efficiency and flexibility. For example, consider stemming the input by converting all forms of a word such as "stemming," "stemmed," and "stemmer" to the root word "stem." This does not reduce the number of stored regions (ignoring for the moment the fact that it can reduce the total space requirement in some compressed representations). However, it reduces the flexibility since it is no longer possible to search for just one of the forms. On the other hand, it increases the efficiency of searching for all forms together since there is no need to combine separate lists. Overall, choosing what regions to pre-compute means predicting the types of queries that need to be supported.

Various representations are possible for persistently stored regions. The simplest is a collection of unordered lists. Given that region lists generated by scanning are ordered by position in the string, however, it makes sense to store them in that order. This can be done in a flat file which allows sequential access or binary searching. Alternatively, it can be in an index structure such as a B-tree which allows more efficient searches and updates. Ordered representations allow the possibility of particularly efficient compression using various techniques.

For a non-nesting set of regions, there is a single, unique sort order. For a set of regions with nesting there are two possible sort orders. That is, regions may ordered primarily by either *s* or by *e*. The choice has some effect on evaluation strategies for expressions that compose multiple operators. If regions are stored persistently, then a single sort order must be chosen. Of course, a physical operator can be provided to convert between the two orderings, although this is not possible in linear time and constant memory. Note that considering such an operator part of the algebra itself implies modifying the definition to use lists of regions with physical order properties rather than sets of unordered regions. Alternatively, one can make a distinction between the logical algebra that works with un-ordered sets, and the physical operators that are used for optimization and evaluation.

One way to avoid choosing between the two possible sort orders in persistent storage is to only index non-nested regions. This is less of a limitation than it might appear since it is easy to generate nested regions dynamically given two lists of endpoints. For example, it is possible to index the start and end tags for nested sections in a text in separate lists. The tags themselves have a unique order since they are not nested. A physical operator can scan and merge the two lists of tag regions, pushing the start tags onto a stack and popping the stack at every end tag. This results in a region list ordered primarily by *e*. A very similar operator can be used to dynamically generate regions that contain two words of interest. In this case, there is no natural pairing of word occurrences and the result may be as big as the cross product of the two lists. A useful solution is to define the operator so that it discards all but the shortest regions in the cross product, or equivalently, those that do not contain another nested region.

Given operands that are stored in an ordered list representation, there are many useful operators such as *contains* and *containedIn* that can be evaluated by scanning both lists and performing a type of merge join. This produces an ordered result that can then be used as an input to another operator. If an algebra consists exclusively of operators that can be evaluated with such merge strategies, then it is possible to evaluate arbitrarily composed expressions with very little query optimization effort and relatively good efficiency. A simple strategy is to evaluate the operator with the two smallest available inputs at each step until all operators in the tree are finished. Intermediate results can be buffered.

There are many ways to improve on this basic strategy. Some of them are implied by viewing the problem as an instance of relational query optimization and have not been explicitly described in the context of region algebras. For example, it is possible to consider additional physical operators such as index nested loop join. This can be used in the above strategy when joining a small operand with a larger one. Rather than scan the entire large operand, the join operator performs a binary search or index lookup within the large operand for each region in the small operand.

It is also possible to consider optimizing an entire query rather than making a local, greedy decision at each step about which operator to evaluate next. This implies the need to determine equivalences between expressions, to consider different physical evaluation plans based on these equivalences, and to choose

between the plans. The choice must involve estimating the costs of physical plans which is a difficult problem in general. However, it may admit heuristic or approximate solutions that do well in practice as is the case with relational query optimization.

There are some inherent limitations of the region model. One has to do with the limited information contained in a region. Given just numeric endpoints, it is easy to tell whether two given regions overlap or nest. However, nothing is known about their relationship with other regions that may exist. For example, given two regions *a* and *b* such that *a* contains *b*, there is no way to tell if *a* directly contains *b* meaning that there is no region *c* in the system such that *a* contains *c* and *c* contains *b*. Ignoring the possibility of constraints that may allow this to be inferred, the only way to know if this is the case is to search every other region list. This is likely to be inefficient.

By definition, operators in a region algebra are also excluded from referring to the content of a region. Thus, it is not possible to define an operator that takes the region endpoints and goes back to the original string to check some condition. Note that this is related to the issue of which regions have been indexed in a string. If there is a set of regions for "stem," an operator cannot check the string to see which of those regions were really the word "stemmed." However, even if there are separate region lists for "stem" and "stemmed," it is not possible to look up the value of words that follow "stem" in the text.

Of course, both of these limitations can be overcome by extending from regions to arbitrary tuples of information. To solve the direct containment problem, it is possible to store a third number *depth* with each region that indicates its nesting level. Then *b* is directly contained in *a* if it is contained in *a* and its depth is higher by one. Similarly, it is possible to add a text column to a region to store the word that follows the region and carry it with subsequent intermediate results. Essentially then, it is possible to consider extending to a full relational model. However, the region columns in such an extended model might admit the possibility of useful physical operators not generally provided in relational systems. For example, many region algebra operators can be executed with list merges but cannot be formulated as joins with equality conditions.

Another general limitation of region algebras has to do with the use of regions as a structure model. Consider a collection of region sets where each set has an associated type or label. This can represent arbitrarily nested structures in a strictly non-overlapping hierarchy. It can also represent multiple independent hierarchies over the same string. It can represent overlapping structures which goes beyond what is possible with a strict hierarchy. One thing that it cannot do is represent an arbitrary graph structure. For example, if regions are mapped to nodes in a graph, and there is an edge between any two nodes that overlap, then not all graphs are possible. For example, this can represent a chain of nodes with edges between them, but cannot add an edge from the first node in the chain to the last node without adding other edges at the same time. This limitation applies even if using regions only to encode a structure without requiring that the endpoints correspond to physical locations in a contiguous string.

Key Applications

A region algebra can be used as a query language for searching structured or semi-structured text, for searching structured or semi-structured data encoded in a text format such as SGML or XML, or for information retrieval. Alternatively, it can be used as an underlying model for query optimization or execution for some other query language.

As a query language, a region algebra can also serve as a component of a larger text processing task. One example is structure recognition where features such as composability and a loose structure model give advantages over alternatives such as grammars [11]. Other examples include constraint definition, outlier finding, and editing by example [9].

Cross-references

- ▶ Information Retrieval
- ▶ Relational Calculus
- ▶ Semi-Structured Query Languages
- ▶ XML
- ▶ XML Indexing

Recommended Reading

1. Burkowski F.J. Retrieval activities in a database consisting of heterogeneous collections of structured text. In Proc. 15th Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1992, pp. 112–125.
2. Burkowski F.J. An algebra for hierarchical organized text-dominated databases. Inform. Process. Manag., 28:313–324, 1994.

3. Clarke C.L.A., Cormack G.V., and Burkowski F.J. An algebra for structured text search and a framework for its implementation. Comput. J., 38(1):43–56, 1995.
4. Consens M.P. and Milo T. Algebras for querying text regions. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems, 1995, pp. 11–22.
5. Consens M.P. and Milo T. Algebras for querying text regions: expressive power and optimization. J. Comput. Syst. Sci., 57:272–288, 1998.
6. Cormack G.V., Clarke C.L.A., Palmer C.R., and Good R.C. The multitext retrieval system (demonstration abstract). In Proc. 22nd Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1999, p. 334.
7. Jaakkola J. and Kilpelinen P. Using sgrep for querying structured text files. In Proc. SGML Finland 1996, J. Saarela (ed.), 1996, pp. 56–67.
8. Jaakkola J. and Kilpeläinen P. Nested text-region algebra. Technical Report C-1999-2, Department of Computer Science, University of Helsinki, January 1999.
9. Miller R.C. Lightweight Structure in Text. Ph.D thesis, School of Computer Science, Carnegie Mellon University, 2002.
10. Salminen A. and Tompa F. PAT expressions: an algebra for text search. Acta Linguistica Hungarica, 41(1–4):277–306, 1992.
11. Young-Lai M. Text Structure Recognition Using a Region Algebra. Ph.D thesis, Department of Computer Science, University of Waterloo, 2000.
12. Young-Lai M. and Tompa F.W. One-pass evaluation of region algebra expressions. Inf. Syst., 28(3):159–168, 2003.

Region Segmentation

► Image Segmentation

Registration Time

► Transaction Time

Regulatory Compliance in Data Management

Radu Sion
Stony Brook University, Stony Brook, NY, USA

Definition

Regulatory compliance in data management refers to information access, processing and storage mechanisms designed according to regulations governing their respective data types and semantics. For example, in the United States, health-related data falls under the incidence of the Health Insurance Portability and Accountability Act (HIPAA) and any associated data management systems need to provide compliance to HIPAA requirements, including data retention and secure deletion assurances. Such compliance has potential for far reaching impact in the design of data processing semantics, from relational ACID properties to complex query processing and index optimization.

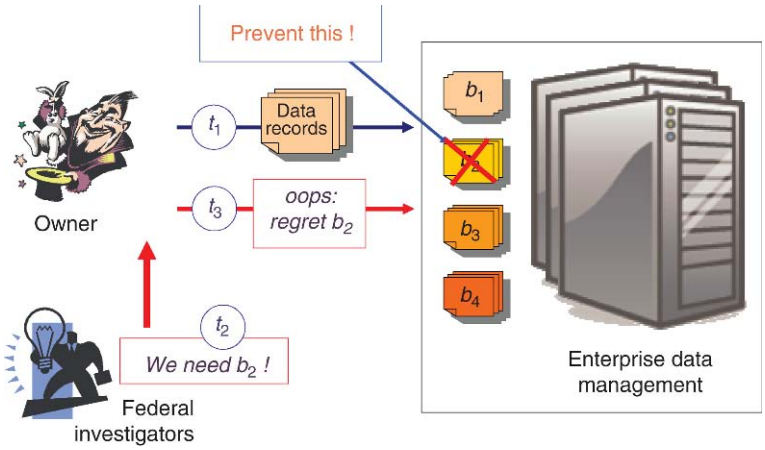
Historical Background

Modern digital societies and markets increasingly mandate consistent procedures for the access, processing and storage of information. In the United States alone, over 10,000 such regulations can be found in financial, life sciences, health-care and government sectors, including the Gramm–Leach–Bliley Act (1999), the Health Insurance Portability and Accountability Act (1996), and the Sarbanes–Oxley Act (2002). A recurrent theme in these regulations is the need for regulatory – compliant data management as an underpinning to ensure data confidentiality, access integrity and authentication; provide audit trails, guaranteed deletion, and data migration; and deliver Write Once Read Many (WORM) assurances, essential for enforcing long-term data retention and life-cycle policies.

Foundations

Overview

Over 10,000 regulations govern the management of documents in the United States alone [8], in financial, life sciences, health-care industries, and the government. These regulations impose a wide range of policies, ranging from information life-cycle management (e.g., mandatory data retention and deletion) to audit trails and storage confidentiality. Examples include the Gramm–Leach–Bliley Act [6], Health Insurance Portability and Accountability Act [13], Federal Information Security Management Act [14], Sarbanes–Oxley Act [15], Securities and Exchange Commission rule 17a-4 [12], Department of Defense Records Management Program under directive 5015.2 [9], Food and Drug Administration 21 CFR Part 11 [11], and the Family Educational Rights and Privacy Act [10]. In Europe, the Directive 95/46/EC of the European Union on the protection of personal data, provides privacy and security guarantees for personal information [3]. In addition, many



Regulatory Compliance in Data Management. Figure 1. WORM prevents history "re-writing".

countries have their own data protection laws. For example, in the United Kingdom, the Data Protection Act of 1998 [1] regulates, among other information, personal health-care records. It requires mandatory disposal of electronic records after retention period, accuracy of information, logging any changes, and strict confidentiality.

While each regulation has its own unique characteristics, certain assurance features can be found throughout:

Guaranteed data retention. To address this requirement, the goal of compliant data management is to support Write Once Read Many (WORM) semantics: once written, data cannot be undetectably altered or deleted before the end of their regulation-mandated life span, even with physical access to its hosting server.

Secure deletion. Once data has reached the end of its lifespan, it can (and in some cases must) be deleted. Deleted records should not be recoverable even with unrestricted access to the underlying medium; moreover, after data is deleted, no hints of its existence should remain on the server, even in the indexes. The term *secure deletion* is used to describe this combination of features.

Compliant data migration. Retention periods are measured in years. For example, national intelligence information, educational records, and certain health records have retention periods of over 20 years. To address this requirement, compliant data management needs *data migration* mechanisms that allow information to be transferred from obsolete to new storage media while preserving its associated security guarantees.

Litigation holds. Even if a data record has reached the end of its lifespan, it should remain fully accessible if it is the subject of current litigation.

In addition to these features, a common thread running through many of these regulations is the perception of powerful insiders as the primary adversary. These adversaries have superuser powers coupled with full access to the storage system hardware. This corresponds to the perception that much recent corporate malfeasance has been at the behest of CEOs and CFOs, who also have the power to order the destruction or alteration of incriminating records. Since the visible alteration or destruction of records is tantamount to an admission of guilt in the context of litigation, a successful adversary must perform their misdeeds *undetectably.*

Regulatory Compliant Commercial Systems

As the ability to control data retention is an essential requirement to be found in a majority of regulations, systems aimed at providing WORM assurances have been brought to the market by virtually all major storage vendors, including companies such as IBM [5], HP, EMC [2], Hitachi Data Systems Zantaz, StorageTek, Sun Microsystem Network Appliance and Quantum Inc. [7]. A set of representative instances are discussed in the following.

Tape-based WORM

Due to the favorable cost-per-MB ratio of tape-based storage in the past, it was a natural choice for massive data storage in commercial enterprise deployments (where regulatory compliance is of concern). Thus

storage vendors started offering tape-based WORM mechanisms first. The Quantum DLTSage predictive, preventative and diagnostic tools for tape storage environments [7] are a representative instance. The WORM assurances of the tape systems are provided under the assumption that only Quantum tape-readers are deployed. "DLTSage WORM provides features to assure compliance, placing an electronic key on each cartridge to ensure WORM integrity. This unique identifier cannot be altered, providing a tamper-proof archive cartridge that meets stringent compliance requirements to ensure integrity protection and full accessibility with reliable duplication." [7]. Such systems however, are subject to a set of impractical assumptions. Given the nature of magnetic tape, an attacker can easily dismantle the plastic tape enclosure and access the underlying data on a different customized reader, thus compromising its integrity. Relying on the physical integrity of a "plastic yellow label" to safe-guard essential enterprise information is likely unacceptable in any medium to high-stake commercial scenarios.

Optical-Disk WORM

Optical (compact) disk (CD) media has been around experimentally since 1969 and commercially available since 1983. Given the prohibitive costs of high-powered lasers in small form factors, in the early days, most CD devices were only capable of reading disk information. As the technology matured, write-once (and later read-write) media appeared. Optical WORM-disk solutions rely on the irreversible physical primitive write effects to ensure the inability to alter existing content. However, with ever increasing amounts of information being produced and requiring constant low-latency accessibility in commercial scenarios, it is challenging to deploy a scalable optical-only WORM solution. Moreover, optical WORM disks are plagued with other practical issues such as the inability to fine-tune WORM and secure deletion granularity (problems partially shared also by tape-based solutions). Moreover, as the WORM disks do not provide any strong security features (due to bulk-production requirements) any optical WORM solutions are vulnerable to simple data replication attacks as discussed above. Optical WORM also perform relatively poorly in the price-performance measurements because current technology is somewhat under-sized for the volumes of data associated with compliance. Sony's Professional Disk for Data optical disk system, for example, holds only 23 GB per disk side. Nevertheless, because it is faster than tape and cheaper than hard disks, optical WORM storage technology is often deployed as a secondary, high-latency storage medium to be used in the framework of a hard disk-based solution. Care needs to be taken in establishing points of trust and data integrity when information leaves the secured hard disk store for the optical media.

Hard Disk-based WORM

Magnetic disk recording currently offers better overall cost and performance than optical or tape recording. Moreover, while immutability is often specified as a requirement for records, what is required in practice is that they be "term-immutable", i.e., immutable for a specified retention period. Thus almost all recently-introduced WORM storage devices are built atop conventional rewritable magnetic disks, with write-once semantics enforced through software ("soft-WORM").

EMC Centera. The EMC Centera Compliance Edition [2] is a content addressed storage (CAS) solution that also offers regulatory compliance capabilities. Each data record "has two components: the content and its associated content descriptor file (CDF) that is directly linked to the stored object (business record, e-mail, etc.). A digital fingerprint derived from the content itself is the content's locator (content address). The CDF contains metadata record attributes (e.g., creation date, time, format) and the object's content address. The CDF is used for access to and management of the record. Within this CDF, the application will assign a retention period for each individual business record. Centera will permit deletion of a pointer to a record upon expiration of the retention period. Once the last pointer to a record has been so deleted, the object will be eliminated" [2], and, in the Plus version, also "shredded" (from the media). Given its software-only nature, these mechanisms are vulnerable to simple software or physical direct disk-access attacks like described above. Data integrity can be easily compromised.

Hitachi message archive for compliance. Similarly, Hitachi Data Systems provide the Data Retention Utility, a software-based "virtual" WORM mechanism for mainstream Hitachi storage systems. The system allows customers to "lock down archived data, making it non-erasable and non-rewritable for prescribed

periods, facilitating compliance with governmental or industry regulations".

IBM LockVault compliance software. IBM offers multiple soft-WORM solutions. The LockVault compliance software is a layer that operates on top of IBM System Storage N series to provide "disk-based regulatory compliance solutions for unstructured data".

IBM system storage archive manager. The IBM Tivoli Storage Manager is part of the IBM TotalStorage Software [5] and provides certain software data retention protection. It "makes the deletion of data before its scheduled expiration extremely difficult. *Short of physical destruction to storage media or server, or deliberate corruption of data or deletion of the Archive Manager database*, Archive Manager will not allow data [...] to be deleted before its scheduled expiration date." However, it is not desirable to base the functioning of the regulatory compliance mechanism on the correct behavior of the main system. After all, the compliance mechanism's main role is to guarantee exactly such fault-less behavior. The main adversary of concern in regulatory settings is *exactly* one with incentives for data corruption and physical compromise attacks.

Network appliance snaplock compliance/enterprise software. The NetApp SnapLock software suite is designed to work on top of NetApp NearStore and FAS storage systems. It provides soft-WORM assurances, "preventing critical files from being altered or deleted until a specified retention date". As opposed to other vendors, NetApp SnapLock supports open industry standard protocols such as NSF and CIFS.

Sun StorageTek compliance archiving software. Sun also offers soft-WORM assurances through its StorageTek Compliance Archiving Software . The software runs on top of the Sun StorageTek 5320 NAS Appliance to "provide compliance-enabling features for authenticity, integrity, ready access, and security".

Security Properties

The design of compliance data management is extremely challenging due to the conflict between security, cost-effectiveness, and efficiency. For example, the requirement to find requested information quickly means in practice data must be indexed. But trustworthy indexing of compliance data is a challenging problem, as it is easy to tamper with traditional indexes stored on WORM. Further, trustworthy indexes will make it very hard to delete all traces of documents that are past their retention periods, as required by certain regulations. Yet another complicating factor is the decades-long retention periods required by many regulations; it is unrealistic to expect data to reside on the same device for so long. One of the main challenges of such an endeavor lies in securing the system against attack by insiders with superuser powers, and in balancing the conflicting requirements for trustworthiness, high performance, and low cost.

Current regulatory compliant data management systems constitute an important step in the right direction. These systems however, are unfortunately fundamentally vulnerable to faulty behavior or illicit adversaries with incentives to alter stored data, as they rely on enforcement primitives – such as software and/or simple hardware device-hosted on/off switches – ill-suited to their target (insider) adversarial setting.

Achieving a secure, cost-effective, and efficient design when the insider is the adversary is extremely challenging. To defend against insiders, processing components that are both tamper-resistant and *active*, such as general – purpose trustworthy hardware are needed. By offering the ability to run logic within a secured enclosure, such devices allow fundamentally new paradigms of trust. Trust chains spanning untrusted and possibly hostile environments can now be built by deploying secure tamper-resistant hardware at the storage components' site. The trusted hardware can run certified logic; close proximity to data coupled with tamper-resistance guarantees allow an optimal balancing and partial decoupling of the efficiency/security trade-off. Assurances can now be both efficient and secure.

However, trusted hardware devices are not a panacea. Their practical limitations pose a set of significant challenges in achieving sound regulatory-compliance assurances. Specifically, heat dissipation concerns under tamper-resistant requirements limit the maximum allowable spatial gate-density. As a result, general-purpose secure coprocessors (SCPUs) are often significantly constrained in both computation ability and memory capacity, being up to one order of magnitude slower than host CPUs. Such constraints mandate careful consideration in achieving efficient protocols. Direct implementations of the full processing logic *inside* the SCPU are bound to fail in practice due to lack of performance. The server's main CPUs will remain starkly under-utilized and the entire cost-proposition of having fast untrusted main CPUs and expensive slower secured CPUs will be defeated. Efficient

protocols need to access the secure hardware sparsely, asynchronously from the main data flow.

Recently, Hsu et al. [4] analyzed some of the principal requirements for trustworthy "content immutable storage" (CIS) of electronic records and identified a set of principles, including: (i) increasing the cost and conspicuity of any attack against the system, (ii) focusing on end-to-end trust, rather than single components, (iii) using a small trusted computing base, isolate trust-critical modules and make them simple, verifiable and correct, (iv) using a simple, well-defined interface between trusted and untrusted components, and (v) trust, but verify every component and individual operation.

Key Applications

Recent compliance regulations are intended to foster and restore humans trust in digital information records and, more broadly, in our businesses, hospitals, and educational enterprises. As increasing amounts of information are created and live digitally, compliance data management will be a vital tool in restoring this trust and ferreting out corruption and data abuse at all levels of society.

Future Directions

Future research will need to explore novel regulatory compliant data management solutions that address the appropriate insider adversary. Numerous technical challenges are associated with such an endeavor, including designs for SCPU overhead-minimizing techniques for expected transaction loads, amortized commitment schemes to enforce write-once-read-many (WORM) semantics at the throughput rate of the servers ordinary processors, while benefiting from existing work on compliance indexing to create efficient indexes secured by novel cryptographic constructs such as fast short-lived signatures. Compliant data migration mechanisms will likely require backwards-compatible, inter-device trust chains and fast re-encryption techniques.

Cross-references

► Trusted Hardware

Recommended Reading

1. British Parliament. Data Protection Act of 1998. http://www.staffs.ac.uk/legal/privacy/dp10rules/index.php, 1998.
2. EMC. Centera Compliance Edition Plus. http://www.emc.com/centera/ and http://www.mosaictech.com/pdf_docs/emc/centera.-pdf, 2007.
3. European Parliament. Legislative Documents. Online at http://ec.europa.eu/justice_home/fsj/privacy/law/index_en.htm, 2006.
4. Hsu W., Huang L., and Ong S. Content Immutable Storage: Truly Trustworthy and Cost-Effective Storage for Electronic Records. Research Report RJ 10332. Technical Report, 2004.
5. IBM Corp. IBM TotalStorage Enterprise. http://www03.ibm.com/servers/storage/, 2007.
6. National Association of Insurance Commissioners. Graham-Leach-Bliley Act, 1999. www.naic.org/GLBA.
7. Quantum Inc. DLTSage Write Once Read Many Solution. http://www.quantum.com/Products/TapeDrives/DLT/SDLT600/DLTIce/Index.aspx and http://www.quantum.com/pdf/DS00232.pdf, 2007.
8. The Enterprise Storage Group. Compliance: The effect on information management and the storage industry. Online at http://www.enterprisestoragegroup.com/, 2003.
9. The U.S. Department of Defense. Directive 5015.2: DOD Records Management Program. Online at http://www.dtic.mil/whs/directives/corres/pdf/50152std_061902/p50152s.pdf, 2002.
10. The U.S. Department of Education. 20 U.S.C. 1232g; 34 CFR Part 99:The Family Educational Rights and Privacy Act (FERPA). http://www.ed.gov/policy/gen/guid/fpco/ferpa, 1974.
11. The U.S. Department of Health and Human Services Food and Drug Administration. 21 CFR Part 11: Electronic Records and Signature Regulations. http://www.fda.gov/ora/compliance_ref/part11/FRs/background/pt11finr.pdf, 1997.
12. The U.S. Securities and Exchange Commission. Rule 17a-3&4, 17 CFR Part 240: Electronic Storage of Broker-Dealer Records. Online at http://edocket.access.gpo.gov/cfr_2002/aprqtr/17cfr240.17a-4.htm, 2003.
13. U.S. Department of Health & Human Services. The Health Insurance Portability and Accountability Act (HIPAA), 1996. www.cms.gov/hipaa.
14. U.S. Public Law 107-347. The E-Government Act, 2002.
15. U.S. Public Law No. 107-204, 116 Stat. 745. newblock The Public Company Accounting Reform and Investor Protection Act, 2002.

Re-identification

► Record Linkage

Re-Identification Risk

► Disclosure Risk

Relational Algebra

Val Tannen
University of Pennsylvania, Philadelphia, PA, USA

Definition

The operators of the relational algebra were already described in Codd's pioneering paper [2]. In [3] he introduced the term *relational algebra* and showed its equivalence with the tuple relational calculus.

This entry details the definition of the relational algebra in the *unnamed perspective* [1], with selection, projection, cartesian product, union and difference operators. It also describes some operators of the *named perspective* [1] such as join.

The flagship property of the relational algebra is that it is equivalent to the (undecidable!) set of domain independent relational calculus queries thus providing a standard for *relational completeness*.

Key Points

Fix a countably infinite set $\mathbb{D}$ of constants over which Σ-*instances* are defined for a relational schema Σ.

The relational algebra is a *many-sorted* algebra, where the sorts are the natural numbers. The idea is that the elements of sort n are finite n-ary relations. The *carrier* of sort n of the algebra is the set of finite n-ary relations on $\mathbb{D}$ If f is a many-sorted k-ary operation symbol that takes arguments of sorts $n_1,...,n_k$ (in this order) and returns a result of sort n then its type is written as follows: $f : n_1 \times ... \times n_k \to n_0$, and this is simplified to n for nullary ($k = 0$) operations. Bold letters $\mathbf{x}$, $\mathbf{y}$ used for tuples and x_i for the i'th component of $\mathbf{x}$. The operations of the algebra, with their types and their interpretation over the relational carriers are the following:

constant-singletons $\{c\} : 1 (c \in \mathbb{D})$.

selection1 $\sigma^n_{ij} : n \to n$ $(1 \leq i < j \leq n)$ interpreted as $\sigma^n_{ij}(R) = \{\mathbf{x} \in R \mid x_i = x_j\}$

selection2 $\sigma^n_{ic} : n \to n$ $(1 \leq i \leq n, c \in \mathbb{D})$ interpreted as $\sigma^n_{ic}(R) = \{\mathbf{x} \in R | x_i = c\}$

projection $\pi i^n_1 ... ik : n \to k$ $(1 \leq i_1,...,i_k \leq n$, not necessarily distinct) interpreted as $\pi^n_{i_1 ... i_k}(R) = \{x_{i_1},..., x_{i_k} \mid \mathbf{x} \in R\}$

cartesian(cross-) product $\times^{mn} : m \times n \to m + n$ interpreted as $\times^{mn}(R, S) = \{x_1,...,x_m, y_1,...,y_n \mid \mathbf{x} \in R \wedge \mathbf{y} \in S\}$

union $\cup^n : n \times n \to n$ interpreted as $\cup^n(R, S) = \{\mathbf{x} | \mathbf{x} \in R \vee \mathbf{x} \in S\}$

difference $-^n : n \times n \to n$ interpreted as $-^n(R, S) = \{\mathbf{x} | \mathbf{x} \in R \wedge \mathbf{x} \notin S\}$

Relational algebra *expressions* are built, respecting the sorting, from these operation symbols, using the relational schema symbols *as variables*.

Note that an obvious operation, intersection, is missing. Of course, intersection can be defined from union and difference, by De Morgan's laws. Interestingly, intersection is also definable just from cartesian product, selection, and projection. Other useful operations are definable also.

Given a relational schema Σ, a relational algebra *query* is an algebraic expression constructed from the symbols in Σ and the relational algebra operation symbols. Given a database instance $\mathcal{I}$ as input, such a query e returns a relation $e(\mathcal{I})$ as output. For example if R, S are binary, the expression $\pi_{2414}(\sigma_{13}(R \times S)) - (R \times R)$ defines a query that returns a 4-ary relation (omit the operation's superscripts because they can usually be reconstructed and use infix notation for the binary operations). Clearly, each of the operations of the relational algebra maps finite instances to finite relations, and more importantly, it is easily seen that *relational algebra queries are domain independent*. Moreover, there exists an (easily) computable translation that takes any relational algebra query into an equivalent domain independent FO (first-order) query.

The converse of this last fact is the main "raison d'être" for the relational algebra. However, because the set of domain independent FO queries is not decidable, it is not possible to define an effective translation just for these queries. Instead, define a "translation" for *all* FO queries, such that domain independent FO queries are indeed translated to equivalent relational algebra queries. Recalling the notation $q(\mathcal{I}/D)$ gives

Theorem

There exists a total computable translation that takes any FO query q into a relational algebra query e such that for any instance $\mathcal{I}$, gives $e(\mathcal{I}) = q(\mathcal{I}/adom(\mathcal{I}) \cup adom(q))$ (and for domain independent queries the right-hand side further equals $q(\mathcal{I})$.

The key to the proof is the observation that active domains can be computed in the relational algebra.

This result justifies Codd calling a query language *relationally complete* whenever it has the expressive

power of the relational algebra. However, it is necessary to note that the relational algebra also inherits the negative results about first-order logic: it is undecidable whether there exists some instance on which a given query returns a non-empty answer (satisfiability) and it is undecidable whether two queries are equivalent.

The presentation above assumes that the relational symbols have just arity. This is the so-called *unnamed perspective* [1] and it is convenient for theoretical investigations. The practical descriptions of the relational model, e.g., [4], use the *named perspective*[1] in which a set of *attributes* is fixed and each relation is organized vertically by a finite set of them. For such as relation, a tuple is function from its attributes to constants. The relational algebra operators in the named perspective are similar to the ones above except that attributes are used instead of the integers identifying components of tuples. Complications arise with cartesian product when the two relations have attributes in common. Thus, in the named perspective an additional operator is needed for the *renaming* of a relation's attributes. On the positive side, using attributes leads to an nice generic definition of *natural join*, and operation that generalizes both cartesian product (when attribute sets are disjoint) and intersection (when attribute sets are identical) and which has many elegant properties.

Cross-references

- ▶ Cartesian Product
- ▶ Computationally Complete Relational Query Languages
- ▶ Difference
- ▶ Division
- ▶ First-Order Logic: Semantics
- ▶ First-Order Logic: Syntax
- ▶ Join
- ▶ Natural Join
- ▶ Projection
- ▶ Relational Algebra
- ▶ Selection
- ▶ Union

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases: The Logical Level. Addison Wesley, Reading, MA, 1994.
2. Codd E.F. A Relational Model of Data for Large Shared Data Banks. Commun. ACM, 13(6):377–387, 1970.
3. Codd E.F. Relational completeness of database sublanguages, In Courant Computer Science Symposium 6: Data Base Systems, R. Rustin (ed.). Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 65–98.
4. Ramakrishnan R. and Gehrke J. Database Management Systems, 3rd edn. McGraw-Hill, New York, 2003.

Relational Algebra for XML

- ▶ XML Tuple Algebra

Relational Calculus

Val Tannen
University of Pennsylvania, Philadelphia, PA, USA

Synonyms

Domain relational calculus; Tuple relational calculus; First-order query

Definition

The relational database model was proposed by Codd in [2] where he assumed that its "data sublanguage" would be based on the predicate calculus (FOL) and where he introduced various algebraic operations on relations. Only in [3] did he introduced the terms *relational algebra* and *relational calculus*.

Later, it became customary to talk about the *domain relational calculus* (detailed below), which is closely related to the syntax of first-order logic and has quantified variables ranging over individual constants, and about the *tuple relational calculus* which is in fact the one given by Codd in [3] and whose variables range over tuples of constants. The two calculi are equivalent, via easy back and forth translations. However, both calculi allow the formulation of *domain dependent* queries which are inappropriate for database languages. While domain independence is undecidable, it is possible to define decidable sublanguages of *safe* queries which are themselves domain independent and such that any domain independent query is equivalent to a safe one.

Key Points

A *relational (database) schema* is a finite first-order vocabulary consisting only of relation symbols. In fact, relational formalisms also permit constants so it is necessary to fix a countably infinite set $\mathbb{D}$ of constants and work with formulae over the first-order

vocabulary $\Sigma \cup \mathbb{D}$ where Σ is a relational schema. Also, the set $\mathbb{D}$ is taken as the sole *universe of discourse* for the interpretation of formulae. A *relational (database) instance* for a given schema Σ (a Σ-*instance*) is a first-order structure whose domain, or universe, is $\mathbb{D}$ and in which the relation symbols are interpreted by *finite* relations, while the constants are interpreted as themselves.

For a given schema Σ , a *domain relational calculus query* (a.k.a. first-order query) has the form $\{\langle e_1,...,e_n\rangle \mid \varphi\}$ where $e_1,...,e_n$ are (not necessarily distinct) variables or constants and φ is a first-order formula over the vocabulary $\Sigma \cup \mathbb{D}$ with equality such that the all free variables of φ occur among $e_1,...,e_n$. The *inputs* of the query are the Σ-instances. For each input $\mathcal{I}$, the *output* of the query $q \equiv \{\langle e_1,...,e_n\rangle \mid \varphi\}$ is the n-ary relation

$$q(\mathcal{I}) = \{\langle \bar{\mu}(e_1),..., \bar{\mu}(e_n)\rangle \mid \text{assignment } \mu \text{ such that } \mathcal{I}, \mu \models \varphi\}$$

Let $\mathcal{I}$ be an instance. The *active domain* of $\mathcal{I}$, notation $adom(\mathcal{I})$, is the set of all elements of $\mathbb{D}$ that actually appear in the relations that interpret Σ in $\mathcal{I}$. While $\mathbb{D}$ is infinite, $adom(\mathcal{I})$ is always finite. Moreover, given a query $q \equiv \{\langle e_1,...,e_n\rangle \mid \varphi\}$ $adom(q)$ is denoted by the (finite) set of constants that occur in φ or among $e_1,...,e_n$. One expects that the instance $\mathcal{I}$ together with the query q completely determines the output $q(\mathcal{I})$. In particular, only the elements in $adom(\mathcal{I}) \cup adom(q)$ can appear in the output. However, this is not the case for all first-order queries. For example, the outputs of $\{x \mid \neg R(x)\}$ or $\{\langle x,y\rangle \mid R(x) \vee S(y)\}$ are in fact infinite! More subtly, the following query is also problematic: $\{x \mid \forall y R(x,y)\}$. Here the output contains only elements from $adom(\mathcal{I})$ but whether a tuple is in the output or not depends on the set of elements that y ranges over. These queries are "dependent on the domain." More precisely, for any instance $\mathcal{I}$ and any D such that $adom(\mathcal{I}) \cup adom(q) \subseteq D \subseteq \mathbb{D}$, denote by $q(\mathcal{I}/D)$ the output of the query q on the input structure obtained by restricting the domain to D. A query q is *domain independent* if for any $\mathcal{I}$ and any D_1,D_2 where $adom(\mathcal{I}) \cup adom(q) \subseteq D_i \subseteq \mathbb{D}, i = 1, 2$ one has $q(\mathcal{I}/D_1) = q(\mathcal{I}/D_2)$. It is generally agreed that in a reasonable *query language*, all the queries should be domain independent. Therefore, general first-order queries do not make a good query language. Worse, it is undecidable whether a first-order query is domain independent [1]. So how does one get a reasonable query language? It is possible (in several ways) to define decidable *safety* restrictions on general first-order formulae such that the safe queries are domain independent and moreover for any domain independent query there exists an equivalent safe query [4,1]. The safety restrictions tend to be complicated and have little practical value. A better idea is the *relational algebra*.

Clearly, not all functions that map instances to relations are meanings of first-order queries. However, the meanings of first-order queries are all *generic*, i.e., invariant under bijective renamings of the constants in $\mathbb{D} \setminus adom(q)$. Conversely, a function f that maps instances to relations and is generic is said to be *first-order definable* (a.k.a. definable in the relational calculus) if there exists a first-order query q whose semantics is f (contrast this with the definability within a given structure described in FIRST-ORDER LOGIC: Semantics). A well-known example of function that is not first-order definable is taking the *transitive closure* of a binary relation.

Codd's *tuple relational calculus* [4,3] differs from the domain relational calculus in that its variables range over tuples and its terms are either constants or of the form $t.k$ where t is a variable and k is a positive integer selecting one of the components of the tuple (for example if t is assigned to (a,b,c) then the meaning of $t.2$ is b).

Cross-references

- ▶ Computationally Complete Relational Query Languages
- ▶ First-Order Logic: Semantics
- ▶ First-Order Logic: Syntax
- ▶ Relational Algebra

Recommended Reading

1. Abiteboul S., Hull R., and Vianu V. Foundations of Databases: The Logical Level. Addison Wesley, Reading, MA, 1994.
2. Codd E.F. A relational model of data for large shared data banks. Commun. ACM, 13(6):377–387, 1970.
3. Codd E.F. Relational Completeness of Database Sublanguages. In Courant Computer Science Symposium 6: Data Base Systems, R. Rustin (ed.). Prentice-Hall, Englewood Cliffs, NJ, 1972, pp. 65–98.
4. Ullman J.D. Principles of Database and Knowledge-Base Systems Volume, I. Computer Science Press, Rockville, MD, 1988.

Relational Database

- ▶ Relational Model

Relational Integrity Constraints

► FOL Modeling of Integrity Constraints (Dependencies)

Relational Model

David W. Embley
Brigham Young University, Provo, Utah, USA

Synonyms

Relational database

Definition

The *Relational Model* describes data as named relations of labeled values. For example, customer ID's can relate with customer names and addresses in the relational model as *Customer*: {<(*CustomerID*, *11111*), (*Name*, *Pat*), (*Address*, *12 Maple*)>, <(*CustomerID*, *22222*), (*Name*, *Tracy*), (*Address*, *44 Elm*)>}. In this example, there is a name for the relation – *Customer*; label-value pairs – e.g., (*CustomerID*, *11111*), which provide the labeled values; and tuples – e.g., <(*CustomerID*, *11111*), (*Name*, *Pat*), (*Address*, *12 Maple*)>, which are the tuples of the named relation.

Usually, the relations of the relational model are viewed as tables. Figure 1 shows an example of several relations viewed as tables. Together, they constitute a relational database. The first table in Fig. 1 is the table view of the relation described in the previous paragraph.

Besides their structures, tables in the relational model also have constraints. Typical constraints include key constraints (e.g., *CustomerID* values must be unique), type constraints (e.g., *DateOrdered* values must be of type *Date*), and referential integrity constraints (e.g., the *CustomerID* values in table *Order* must refer to existing *CustomerID* values in table *Customer*).

Historical Background

Codd's seminal paper [2] introduced the relational model as a model based on *n*-ary relations. The seminal paper also introduces normal forms for relations and a query language for relations. In perspective, Codd essentially showed how to apply fundamental concepts in set theory to form the relational data model, the foundation of relational databases.

Several textbooks describe the relational data model. Widely read descriptions are in the textbook by Elmasri and Navathe [3] and the textbook by Silberschatz, Korth, and Sudarshan [5], both of which appear in their fifth edition. A few textbooks focus solely on the relational model and treat it from a theoretical perspective [1,4].

Foundations

The relational model can be viewed in several ways: (i) intuitively, (ii) conceptually, (iii) as an implementation description, and (iv) formally. Each view satisfies different user needs.

Intuitive View

Figure 1 shows three tables that together model a simple item-order application. An intuitive description of the relational model has the following characteristics:

- A group of tables stores the data of an application, and together the tables constitute a relational database (e.g., the tables in Fig. 1).

Customer

CustomerID	Name	Address
11111	Pat	12 Maple
22222	Tracy	44 Elm

Item

ItemNr	Description
X11-222	Ball
Y33-444	Game
Z11-555	Book

Order

OrderNr	CustomerID	ItemNr	DateOrdered	NrOrdered
123-1	11111	Y33-444	12 May	1
114-7	22222	X11-222	10 May	5
114-7	22222	Y33-444	10 May	5

Relational Model. Figure 1. Sample named relations of labeled data viewed as tables in the relational model.

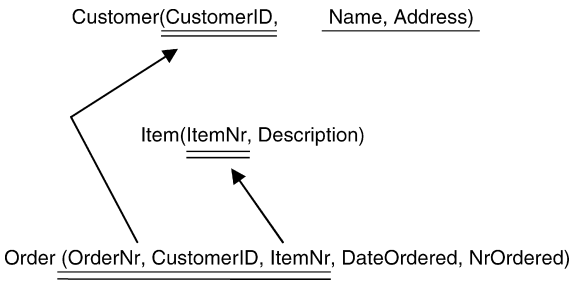
- Each table has a name (e.g., *Customer* for the first table in Fig. 1).
- Each table has labeled column headers (e.g., *CustomerID*, *Name*, and *Address* for the first table in Fig. 1).
- Each table has data for each column header (e.g., *Pat* and *Tracy* for the *Name* column of the first table in Fig. 1).
- The rows of each table constitute a record for the table (e.g., <*11111*, *Pat*, *12 Maple*> for the first row in the first table in Fig. 1).
- Each record asserts a fact that relates the table name, column headers, and record values (e.g., the first row of the first table in Fig. 1 asserts that a *Customer* exists whose *CustomerID* is *11111*, whose *Name* is *Pat*, and whose *Address* is *12 Maple*).
- Each value belongs to a specified domain, a defined set of possible values (e.g., *CustomerID* values are five-digit numbers).
- A key uniquely identifies each record (e.g., in Fig. 1 *CustomerID* uniquely identifies customers and the triple *OrderNr-CustomerID-ItemNr* uniquely identifies a line item in an order).
- Values link records in different tables (e.g., in Fig. 1 the value *11111* links the customer whose *CustomerID* is *11111* with the orders for that customer by recording the *CustomerID* in the same record as the *OrderNr*).
- For these cross-table linkages to make sense, referenced values must exist (e.g., in Fig. 1 an order record references customers by *CustomerID*'s; thus, any *CustomerID* value recorded in the *Order* relation must exist in the *Customer* relation).

Conceptual View

In a conceptual view of the relational model, the focus is on the conceptual schema of each table. The collection of table schemas for all the tables constitutes the relational database schema.

Figure 2 shows a schematic view of the relational database schema for the database in Fig. 1. A *relational schema* is a named template for a set of *n*-tuples with the following features.

- The schema has a name – *Customer*, for example, in Fig. 2.
- The schema has *n* attribute names, one for each component of the *n*-tuples being represented. The *n* attribute names must all be different so that they form a set. The attribute names are usually just called attributes. In Fig. 2 the *Customer* schema has three attributes: *CustomerID*, *Name*, and *Address*.
- The schema has key constraints. An attribute (or a set of attributes) whose value (or values) uniquely determine at most one tuple is a key. One of these keys is the primary key; often there is just one key, which therefore is the primary key. In the schematic view, underlines designate keys; double underlines designate primary keys. In the *Customer* schema in Fig. 2, *CustomerID* is the primary key, and the pair *Name-Address* is another key.
- The schema has referential integrity constraints. This prevents referencing objects that do not exist. In the example in Fig. 2, the customers referred to in the *Order* table must exist in the *Customer* table, and the items referred to must exist in the *Item* table. Almost always, the referring attribute refers to a key, usually the primary key. Thus, the referring attribute is called a foreign key – a key in another table. When an attribute is a foreign key, the values for the attribute in the *n*-tuples for the table must be a subset of the values for the attribute in the *m*-tuples of the referenced table. Schematically, as Fig. 2 shows, an arrow is drawn from a foreign key to a key. In the example, the customer IDs in the *Order* tuples must be a subset of the customer IDs in the *Customer* tuples, and the item numbers in the order tuples must be a subset of the item numbers in the *Item* tuples.
- Optionally, the schema may also have domain specifications for the attributes. For example, customer IDs may always be five-digit numbers. If so, "*CustomerID*: *five-digitnumber*" could be written as a note in the schematic diagram in Fig. 2.
- Optionally, the schema may also have additional information. Additional constraints or notes may



Relational Model. Figure 2. Schematic of table headers.

be written – for example, a note to say that the *DateOrdered* should be initialized as today's date.

Implementation View

The language used to express implementation views is called a Data-Definition Language (DDL). Figure 3 shows an example for the sample customer-order application in Fig. 1. The DDL in Fig. 3 is SQL, a standard commercial database language for specifying, updating, and querying a database.

SQL allows database developers to declare relations for the database and constraints over and among these relations. The SQL syntax provides a way to declare these basic relational-model features as follows.

- A developer declares a schema for a table with a *create table* declaration. The developer must provide a name for the table, and then must declare each of the table's attributes. *Customer* is the name for the first table declared in Fig. 3.
- A developer declares an attribute by giving its name and then listing the constraints that apply to the attribute. The attributes in the *Customer* table declared in Fig. 3 are *CustomerID*, *Name*, and *Address*.
- A developer declares domain constraints for an attribute by giving a type declaration. In Fig. 3 the type declaration for *CustomerID* is *numeric* (5), declaring that customer IDs are 5-digit numbers. SQL provides various types such as numbers, strings, dates, time, and money. Type declarations are unfortunately not uniform across all database systems.

```
create table Customer (
        CustomerID numeric(5) primary key,
        Name varchar(20),
        Address varchar(25),
        unique (Name, Address)
        );

create table Item (
        ItemNr char(7) primary key,
        Description varchar(50)
        );

create table Order (
        OrderNr varchar(10),
        CustomerID numeric(5) references Customer,
        ItemNr char(7) references Item,
        DateOrdered date,
        NrOrdered smallint,
        primary key (OrderNr, CustomerID, ItemNr ),
        );
```

Relational Model. Figure 3. SQL implementation schemas.

- A developer declares key constraints for an attribute either by stating that it is the *primary key* or that it is *unique* – a key but not the primary key. In Fig. 3 *CustomerID* in the first table schema and *ItemNr* in the second table schema are primary keys.
- A developer declares foreign-key constraints for an attribute with a *references* clause. The *references* clause designates the table in which the referenced attribute is found. When the attribute in the referenced table has the same name as the attribute in the referencing table, this simple declaration is sufficient. If the name is different, then the *references* clause must also include the name of the attribute being referenced. In Fig. 3, *CustomerID* in the *Order* table references *CustomerID* in the *Customer* table, and *ItemNr* references *ItemNr* in the *Item* table. These foreign-key constraints ensure that the *CustomerID* and *ItemNr* values in the *Order* table refer to existing values in the *Customer* and *Order* tables.
- When constraints involve multiple attributes, SQL provides syntax that allows a developer to declare these constraints in a separate entry in a table declaration. Thus, as Fig. 3 shows, a developer can declare that the attribute combination consisting of *Name* and *Address* constitutes a key for the *Customer* table, and that the attribute combination consisting of *OrderNr*, *CustomerID*, and *ItemNr* constitutes the primary key for the *Order* table. The SQL syntax also provides for multiple-attribute foreign keys. Thus, although neither necessary nor even desirable in the example in Fig. 3, the attributes *Name* and *Address* could be added to the *Order* table and a foreign-key constraint could then be declared as *foreign key* (*Name*, *Address*) *referencesCustomer* (*Name*, *Address*).
- Typical additional constraints declarable with SQL include null constraints and check constraints. Null constraints let developers decide whether null values can or cannot appear as values for attributes. Values for primary-key attributes may never be null; other attributes require a *not null* designation (otherwise they can have null values). Check constraints let developers add conditions that must hold. For example, a developer can declare that *NrOrdered* can be neither negative nor zero by adding the constraint *check*($NrOrdered > 0$) to the *NrOrdered* attribute in the *Order* table in Fig. 3.

Formal View

The formal view of the relational model captures the essence of a relational schema in terms of mathematical concepts. The definition is based on the concepts of sets, relations, and functions.

A *relational schema* R is a non-empty set of attribute names $R = \{A_1, A_2,...,A_n\}$. Usually an "attribute name" as just called an "attribute." Further, as a notational convenience, when an attribute name is a single letter, the set notation is reduced by dropping the braces, commas, and spaces. Thus, a *relational schema* R is a non-empty set of attributes $R = A_1A_2...A_n$. Each attribute A has a domain, denoted $dom(A)$, which is a set of values.

A relation is always defined with respect to a relational schema. The notation $r(R)$ denotes that relation r is defined with respect to a relational schema R and is read "r is a relation on schema R" or just "r on R" when the context is clear. A *relation* is a set of n-tuples, $\{t_1,..., t_k\}$, where n is $|R|$, the cardinality of R. Let schema R be $A_1A_2...A_n$. Then, an *n-tuplet* for a relation $r(R)$ is a function, from R to the union of domains $D = dom(A_1) \cup dom(A_2) \cup ... \cup dom(A_n)$, with the restriction that $t(A_i) \in dom(A_i)$, $1 <= i <= n$.

A relation is usually written as a table – the table in Fig. 4, for example. In Fig. 4 the relation r has relational schema $R = AB$. To declare the attribute domains, assume $dom(A)$ is $\{a, b, c\}$ and $dom(B)$ is $\{0,...,9\}$. The set of n-tuples for r is the set of discrete functions $\{\{(A, a), (B, 1)\}, \{(A, b), (B, 1)\}, \{(A, b), (B, 2)\}\}$.

Another way to view the relational model formally is to view it as an interpretation in first-order logic. In this view of the relational model each relation r, whose schema has n attributes, is an n-place predicate. An *interpretation* for a first-order language consists of (1) a non-empty domain D of values, which under the unique name assumption each represent themselves, and (2) for each n-place predicate, an assignment of *True* or *False* for each possible substitution of n values from D.

As an example, consider the relation in Fig. 4. To see this relation as an interpretation, let $r(x, y)$ be a two-place predicate. The domain D is {a, b, c, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. *True* is assigned to the substitutions $(x = a, y = 1)$, $(x = b, y = 1)$, and $(x = b, y = 2)$ and *False* is assigned to all other substitutions.

r =

A	B
a	1
b	1
b	2

Relational Model. Figure 4. Sample table for illustrating the formal definition.

Most often in this first-order-logic view, the "closed world assumption" is used to conveniently assign *True* and *False* to each substitution. The "closed world assumption" states that whatever is not *True* is *False*, and thus only the *True* facts need to be recorded. This, then, reduces to the equivalent of just giving a table such as the one in Fig. 4 – the rows in the table represents the substitutions for which the predicate is *True* and the only substitutions for which the predicate is *True*.

In a first-order-logic view of the relational model, it is also possible to express constraints. For the table in Fig. 4, for example, the constraint that A should be a key can be expressed by the closed formula $\forall x_1 \forall y_1 \forall x_2 \forall y_2 (r(x_1, y_1) \wedge r(x_2, y_2) \wedge x_1 = x_2 \Rightarrow y_1 = y_2)$. This does not hold for the table in Fig. 4, however, since the second and third tuples have $x_1 = x_2$, but $y_1 \neq y_2$. A check constraint stating that the values in the B column must all be less than 5 can be expressed as $\forall x \, \forall y (r(x, y) \Rightarrow y < 5)$. This constraint does hold in the table in Fig. 4.

For an interpretation for a first-order language, when all the closed formulas hold, the interpretation is said to be a *model*. Database instances in which all constraints hold are therefore models – models of the world they represent.

Key Applications

Relations, stored in relational databases, are widely used in industry. Indeed, their combined usage constitutes a mega-billion-dollar industry. Relational databases range from relatively small databases used as backends to web applications such as "items for sale" to relatively large databases used to store corporate data.

Cross-references

- ▶ Database Normalization
- ▶ Key
- ▶ Relational Algebra
- ▶ Relational Calculus
- ▶ SQL

Recommended Reading

1. Atzeni P. and De Antonellis V. Relational Database Theory, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1993.

2. Codd E.F. A relational model for large shared data banks, Commun. ACM, 13(6)377–487, 1970.
3. Elmasri R. and Navathe S.B. Fundamentals of Database Systems, Fifth Edition. Addison-Wesley, Boston, MA, 2007.
4. Maier D. The Theory of Relational Databases, Computer Science Press, Inc., Rockville, MA, 1983.
5. Silberschatz A., Korth H.F., and Sudarshan S. Database System Concepts, Fifth Edition. McGraw-Hill, New York, 2006.

Relational Query Processor

► Query Processor

Relation-Completeness

► BP-Completeness

Relations with Marked Nulls

► Naive Tables

Relationship of Reliance

► Trust in Blogosphere

Relationships in Structured Text Retrieval

Mounia Lalmas
Queen Mary, University of London, London, UK

Definition

In structured text retrieval, the relationship between text components may be used in ranking components relative to a given query.

Key Points

In a structured text document, there exists a relationship between the document components. In the context of XML retrieval, the relationships between elements are provided by the logical structure of the XML mark-up. An element, unless it is the root element (the document itself), has a parent element, which itself may have a parent element. Similarly, non-leaf elements have children elements, and so on. Considering relationships between elements appears to be beneficial for XML retrieval. For instance, in a collection of scientific articles, it is reasonable to assume that the "abstract" of an article is a better indicator of what the article is about than a "future work" section in the same article. The challenge in XML retrieval is what types of relationship should be considered, and how this information can be used to score elements according to how relevant they are for a given query. In the contextualization approach [1], considering the "root element – element" relationship to rank an element in addition to the element own content has shown to improve retrieval effectiveness [2,3].

Cross-references

► Contextualization
► Logical Structure
► Structure Weight
► XML Retrieval

Recommended Reading

1. Arvola P., Junkkari M., and Kekäläinen J. Generalized contextualization method for XML information retrieval. In Proc. ACM Conf. on Information and Knowledge Management, 2005, pp. 20–27.
2. Mass Y. and Mandelbrod M. Component ranking and automatic query refinement for XML retrieval. In Advances in XML Information Retrieval and Evaluation, LNCS, vol. 3493, Springer, 2005, pp. 73–84.
3. Sigurbjörnsson B., Kamps J., and de Rijke M. An element-based approach to XML retrieval. In Proc. 2nd Int. Workshop of the Initiative for the Evaluation of XML Retrieval, 2003, pp. 19–26.

Relative Operating Characteristic

► Receiver Operating Characteristic (ROC)

Relative Time

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

Definition

Viewing a temporal database as a collection of time-referenced (or timestamped) facts, a time reference in

such a database is called *relative* if its value is dependent of the context. For example, this context can be the current time, *now*, or it can be another instant.

Key Points

The relationship between times can be qualitative (before, after, etc.) as well as quantitative (3 days before, 397 years after, etc.). If quantitative, the relationship is specified using a time span.

Examples: "Mary's salary was raised yesterday," "it happened sometime last week," "it happened within 3 days of Easter, 2005" "the Jurassic is sometime after the Triassic," and "the French revolution occurred 397 years after the discovery of America".

The simplest example of a relative timestamp is a period that starts at a time in the past and extends to *now*, such as (2005, *now*). As the clock ticks, this period gets longer.

Cross-references

- ▶ Absolute Time
- ▶ Now in Temporal Databases
- ▶ Qualitative Temporal Reasoning
- ▶ Temporal Database
- ▶ Time Span

Recommended Reading

1. Jensen C.S. and Dyreson C.E. (eds.). A consensus glossary of temporal database concepts – February 1998 version. In Temporal Databases: Research and Practice, O. Etzion, S. Jajodia, S. Sripada (eds.). LNCS, vol. 1399. Springer, Berlin, 1998, pp. 367–405.

Relevance

Jovan Pehcevski[1], Birger Larsen[2]
[1]INRIA Paris-Rocquencourt, Le Chesnay Cedex, France
[2]Royal School of Library and Information Science, Copenhagen, Denmark

Synonyms

Exhaustivity; Specificity; Topical-hierarchical relevance

Definition

Relevance is the extent to which some information is pertinent, connected, or applicable to the matter at hand. It represents a key concept in the fields of documentation, information science, and information retrieval.

In information retrieval, the notion of relevance is used in three main contexts. Firstly, an algorithmic relevance score is assigned to a search result (usually a whole document) representing an estimated likelihood of relevance of the search result to a topic of request. This relevance score often determines the order in which search results are presented to the user. Secondly, when the performance of information retrieval systems is tested experimentally, the retrieved documents are assessed for their actual relevance to the topic of request by human assessors (topic experts). A binary relevance scale is typically used to assess the relevance of the search result, where the relevance is restricted to be either zero (when the result is not relevant to the user request) or one (when the result is relevant). Thirdly, in experiments involving users (or in operational settings) a broader notion of relevance is often used, with the aim of expressing the degree to which the retrieved documents are perceived as useful in solving the user's search or work task.

In structured text (XML) retrieval, the search result is typically an XML element, and the relevance score assigned by an XML retrieval system again represents an estimated likelihood of relevance of the search result to the topic of request. However, when the results are subsequently assessed for relevance, the binary relevance scale is not sufficient, primarily due to the hierarchical relationships that exist among the elements in an XML document. Accordingly, in XML retrieval one or more relevance dimensions (each with a multi-graded relevance scale) have been used to assess the relevance of the search result.

Key Points

In traditional information retrieval experiments where whole documents are retrieved, a fairly simple notion of relevance may suffice for most purposes [1]. The challenge in XML retrieval is that the relevance assessments must capture not only whether the retrieved elements are relevant, but also how they relate to one another.

The different relevance definitions for XML retrieval have mainly been investigated by the INitiative for the Evaluation of XML Retrieval (INEX). In 2002, the INEX relevance definition comprised two relevance dimensions named *topical relevance* and *component coverage*. This relevance definition has not been used by INEX since then, partly because of the vague terminology used for the names of the two relevance dimensions, and partly because it has been subsequently

shown that the INEX 2002 assessors did not fully comprehend component coverage. In 2003 and 2004, the two INEX relevance dimensions were named Exhaustivity and *Specificity*, which respectively reflect the extent that an element *covers* and is *focussed on* aspects of an information need represented by the topic of request. The two INEX relevance dimensions used four grades to assess the relevance of an element (either its exhaustiveness or its specificity): "none," "marginally," "fairly," and "highly." The grades from each dimension were then combined into a single 10-point relevance scale.

From 2005 onwards, a highlighting assessment procedure is used at INEX to gather relevance assessments for the XML retrieval topics. As a result, the INEX relevance definition was simplified such that only three Exhaustivity values were assigned to a relevant element, while the *Specificity* of the relevant element was measured on a continuous relevance scale and computed automatically as the ratio of highlighted to fully contained text. From 2006 onwards, relevance in INEX is defined only according to the notion of *Specificity*, where the continuous relevance scale is used to assess the relevance of retrieved elements.

The experience of both assessors and users is important when defining relevance in XML retrieval. An interactive track was established at INEX in 2004 to investigate the behavior of users when elements of XML documents (rather than whole documents) are presented as answers. The interactive track was run again at INEX in 2005 and 2006, comprising various tasks and different XML document collections [2].

A *topical-hierarchical* relevance definition was used by the INEX Interactive tracks in 2005 and 2006, which comprises two relevance dimensions and a five-point nominal relevance scale [2,3]. An analysis of the feedback gathered from users participating in the INEX 2005 Interactive track showed that users did not find the five-point scale to be very hard to understand [3]. Furthermore, a mapping between the five-point relevance scale (used by users) and the continuous *Specificity* scale (used by the expert assessors) can easily be established [3], which allows for a better understanding of the definition of relevance in XML retrieval.

Cross-references

- ▶ Evaluation Metrics for structured text retrieval
- ▶ Similarity and Ranking Operations
- ▶ Specificity

Recommended Reading

1. Borlund P. The concept of relevance in IR. J. Am. Soc. Inf. Sci. Technol., 54(10):913–925, 2003.
2. Malik S., Tombros A., and Larsen B. The interactive track at INEX 2006. In Proc. 5th Int. Workshop of the Initiative for the Evaluation of XML Retrievals, 2007, pp. 387–399.
3. Pehcevski J. Relevance in XML retrieval: the user perspective. In Proc. SIGIR 2006 Workshop on XML Element Retrieval Methodology, 2006, pp. 35–42.

Relevance Evaluation of IR Systems

- ▶ Effectiveness Involving Multiple Queries

Relevance Feedback

Ben He
University of Glasgow, Glasgow, UK

Synonyms

RF

Definition

The relevance feedback technique expands the initial query by taking into account relevance assessments on documents by the user. Relevance feedback techniques aim to improve the retrieval accuracy and to better satisfy the user information need.

Key Points

The relevance feedback algorithms may use explicit (e.g., user labels), implicit (e.g., click-through data) or blind (e.g., pseudo relevance documents) relevance information for the feedback purpose.

A classical relevance feedback algorithm based on the Vector Space model was proposed by Rocchio in 1971 [1].

Cross-references

- ▶ Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model
- ▶ Probabilistic Model
- ▶ Query Expansion Models
- ▶ Relevance Feedback for Content-Based Information Retrieval

► Rocchio's Formula
► Web Search Relevance Feedback

Recommended Reading

1. Rocchio J. 1Relevance Feedback in Information Retrieval. Prentice-Hall, Englewood Cliffs, NJ, USA, 1971, pp. 313–323.

Relevance Feedback for Content-Based Information Retrieval

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

Definition

Relevance feedback (RF) [5,2] is an on-line approach which tries to learn the users' intentions on the fly. It leverages users to guide the computers to search for relevant documents. An RF mechanism has two components: a learner and a selector. At every feedback round, the user marks (part of) the images returned by the search engine as *relevant* or *irrelevant*. The learner exploits this information to re-estimate the target of the user. This information is used both quantitatively (retrieving more documents like the relevant documents) and qualitatively (retrieving documents similar to the relevant ones before other documents). With the current estimation of the target, the selector chooses other images that are displayed by the interface of the search engine; then the user is asked to provide feedback on these images during the next round. The process of RF is usually presented as a cycle of activity: an IR system presents a user with a set of retrieved documents; the user indicates those that are relevant; and the system uses this information to produce a modified version of the query. The modified query is then used to retrieve a new set of documents for presentation to the user. This process is known as an iteration of RF which ends until the user is satisfied.

Historical Background

RF is a powerful tool which is traditionally used in text-based information retrieval systems. It was introduced to CBIR during mid-1990s [4], with the intention of bringing the user into the retrieval loop and reducing the "semantic gap" between what queries represent (low-level features) and what the user thinks.

Early approaches assumed the existence of an ideal query point that, if found, would provide the appropriate answer to the user. These approaches belong to the family of query point movement (QPM) methods, for which the task of the learner consists in finding, at every round, a better query point together with a re-weighting of the individual dimensions of the description space.

Recent work on RF often relies on support vector machines (SVM). With SVMs, generally the data is first mapped to a higher-dimensional feature space using a non-linear transform kernel, then is classified in this mapped feature space using maximum margin strategy [6].

Foundations

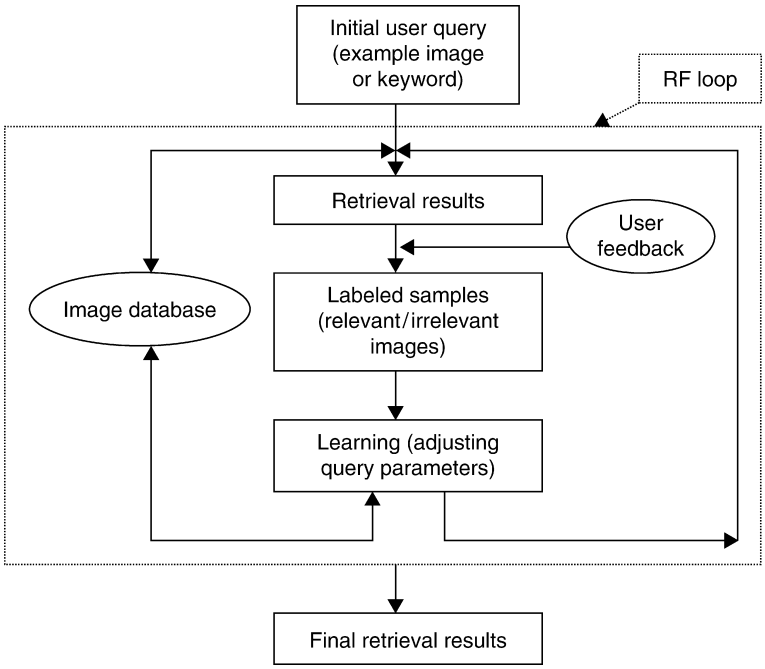
General Assumptions

Before developing relevance feedback mechanisms for content-based image retrieval, it is necessary to understand some general assumptions underlying. According to [2], the assumptions are as the following:

1. The discrimination between relevant and irrelevant images must be possible with the available image descriptors.
2. There are some relatively simple relations between the topology of the description space and the characteristics shared by the images that the user is searching for.
3. Relevant images are a small part of the entire image database.
4. While some of the early work on RF assumed that the user could (and would be willing to) provide a rather rich feedback, including relevance notes for many images, currently the assumption is that this feedback information is scarce. The user will only mark a few relevant images as positive and some very different images as negative.

Figure 1 shows the flowchart of a typical Content-based image retrieval process with relevance feedback [3]. Based on the above general assumptions, a typical scenario for relevance feedback in content-based image retrieval is as below [3,9]:

1. The system provides initial retrieval results given query examples;
2. User judges the above results as to what degree, they are relevant (positive examples) or irrelevant (negative examples) to the query.



Relevance Feedback for Content-Based Information Retrieval. Figure 1. CBIR with RF.

3. Machine learning algorithm is applied to learn a new ranking model based on the user's feedback. Then go back to (2).

Steps (2)–(3) are repeated till the user is satisfied with the results.

Step (3) is comparably the most important step and different approaches can be used to learn the new query. A few generally adopted approached are introduced in the following.

Re-Weighting Approaches

A typical approach in step (3) is to automatically adjust the weights of low-level features to accommodate the users' need, rather than asking the user to specify the weights as adopted in earlier content-based image retrieval systems. This re-weighting step dynamically updates the weights embedded in the query (not only the weights to different types of low-level features such as color, texture, shape, but also the weights to different components in the same feature vector) to model the high-level concepts and perception subjectivity [4].

Query Point Movement Approaches

Another method is called query-point-movement (QPM) [3]. It improves the estimation of the query point by moving it towards the positive examples and away from the negative examples. A widely adopted query point removing technique is called the Rocchio's formula [1] (see (1) below):

$$Q' = \alpha Q + \beta\left(\frac{1}{N_{R'}}\Sigma_{i\in D'_R} D_i\right) - \gamma\left(\frac{1}{N_{N'}}\Sigma_{i\in D'_N} D_i\right) \quad (1)$$

In (1), Q and Q' are the original query and updated query, respectively, and D'_R and D'_N are sets of the positive and negative images returned by the user, and $N_{R'}$ and $N_{N'}$ are the set sizes. α, β and γ are weights.

Machine Learning Approaches

Machine learning techniques are also widely used. As mentioned previously, support vector machine (SVM) is used to capture the query concept by firstly applying the kernel trick which projects images onto a hyperspace and then separates the relevant images from irrelevant ones using maximum margin strategy. The advantages of adopting SVM are that (i) it has high generalization ability, and (ii) it works for small training sets.

Another step-forward approach is proposed by Tong and Chang [8] called SVM active learning, which was reported to be able to effectively use

negative and non-labeled samples, and learn the query concept faster and with better accuracy.

Since manually labeling images is tedious and expensive, training image set is usually very small. This is called the small sample problem. To handle this problem, some researchers proposed boosting methods, e.g., Discriminant-EM (D-EM) [7], which boosts the classifier learnt from the limited labeled training data.

Decision-tree learning methods such as C4.5, ID3 were also used in RF loop to classify the database images into relevant and irrelevant.

Key Applications

Relevance feedback approach can help not only content-based image retrieval, but also applications like image annotation, segmentation, etc.

Future Directions

There are still many research work on adopting relevance feedback to content-based image retrieval [2]:

1. It is better to exploit prior information, such as domain-specific similarity, clustering, context of session, etc., in the RF mechanisms.
2. The impact of the data and of the policy of the user on both the learner and the selector must be addressed.
3. How to scale up RF to handle very large image databases is an important issue which was not extensively studied.

Cross-references

- ► Content-Based Image Retrieval
- ► Relevance Feedback

Recommended Reading

1. Chen Z. and Zhu B. Some formal analysis of Rocchio's similarity-based relevance feedback algorithm. Information Retrieval 5:61–86, 2002.
2. Crucianu M., Ferecatu M., and Boujemaa N. Relevance feedback for image retrieval: a short survey. In state of the art in audiovisual content-based retrieval, Information Universal Access and Interaction, Including Datamodels and Languages. Report of the DELOS2 European Network of Excellence (FP6), (2004).
3. Liu Y., Zhang D., Lu G., and Ma W.-Y. A survey of content-based image retrieval with high-level semantics. Pattern Recognition 40(1):262–282, 2007.
4. Rui Y., Huang T.S., Ortega M., and Mehrotra S. Relevance feedback: a power tool for interactive content-based image retrieval. IEEE Transactions on Circuits and Systems for Video Technology 8(5):644–655, 1998.
5. Ruthven I. and Lalmas M. A survey on the use of relevance feedback for information access systems. The Knowledge Engineering Review(2003). Cambridge University Press, London.
6. Schölkopf B. and Smola A. Learning with Kernels. MIT Press, Cambridge, MA, 2002.
7. Tian Q., Yu Y., and Huang T.S. Incorporate discriminant analysis with EM algorithm in image retrieval. In Proc. IEEE Int. Conf. on Multimedia and Expo, 2000, pp. 299–302.
8. Tong S. and Chang E. Support vector machine active learning for image retrieval. In Proc. 9th ACM Int. Conf. on Multimedia, 2001, pp. 107–118.
9. Zhu X.S. and Huang T.S. Relevance feedback in image retrieval: a comprehensive review. Multimedia System 8(6):536–544, 2003.

Relevance Feedback for Text Retrieval

Olga Vechtomova
University of Waterloo, Waterloo, ON, Canada

Synonyms

RF

Definition

Relevance feedback (RF) is a process by which the system, having retrieved some documents in response to the user's query, asks the user to assess their relevance to his/her information need. The user's relevance judgements are then used to either adjust the weights of the query terms, or add new terms to the query (query expansion).

Key Points

Searchers may have difficulties in finding the words and phrases (terms) to express their information needs accurately and completely. They may also use different words in the queries than the words used by the authors of documents. On the other hand, searchers tend to know relevant information when they see it. In other words, it may be easier for them to tell which documents are relevant, instead of formulating a detailed query.

A typical relevance feedback process consists of the following steps: the user formulates and submits an initial query to an information retrieval system, which retrieves a ranked list of documents. Documents are typically presented in the ranked list in some surrogate

form, for example, as document titles, abstracts, snippets of text, query-biased, or general summaries. The user then reads the contents of the documents that seem promising. After reading each document, the user is expected to indicate to the system whether it is relevant or not to his/her information need. After the user has judged some documents as relevant, either the system, or the user initiate the process of query expansion or query term reweighting. The relevance feedback process can be iterative.

When relevance feedback is used for reweighting, the weights of the user-entered query terms are adjusted based on their occurrence in the judged documents. For example, in the probabilistic model [2], terms are reweighted based on their presence in the documents judged relevant to give a more accurate estimation of the probability of the document's relevance to the query.

Relevance feedback has consistently yielded substantial gains in performance in experimental settings when used for query expansion. In this case, documents judged relevant by the searcher are used as the source of new terms to be added to the query. Many term selection methods have been proposed for query expansion following relevance feedback. The general idea behind all such methods is to select terms that discriminate between relevant and nonrelevant documents, and are useful in retrieving previously unseen relevant documents, e.g., [1,2].

A related process is known as pseudo-relevance or blind feedback (BF). The difference from RF is that instead of asking the user to indicate the relevance of a document, the system assumes that n top ranked documents are relevant. An advantage of BF over RF is that it requires less cognitive effort on the part of the user. The performance of BF, however, depends on the performance of the user's query. If the documents in the top ranks are nonrelevant, then query expansion or query term reweighting will deteriorate the performance.

Another technique is implicit relevance feedback (IRF), where the system infers the user's interest in the documents based on his/her behavior with respect to the IR system. A detailed study analyzing the utility of IRF was conducted by [4].

The adoption of relevance feedback in its traditional form by the web search engines has been limited. A study of the use of relevance feedback in the Excite search engine is reported in [3].

Cross-references

- ▶ Information Retrieval
- ▶ Information Retrieval Models
- ▶ Query Expansion for Information Retrieval

Recommended Reading

1. Carpineto C., de Mori R., Romano G., and Bigi B. An information-theoretic approach to automatic query expansion. ACM Trans. Inf. Syst., 19(1):1–27, 2001.
2. Spärck Jones K., Walker S., and Robertson S.E. A probabilistic model of information retrieval: development and comparative experiments. Inf. Process. Manage., 36(6):779–808 (Part 1); 809–840 (Part 2), 2000.
3. Spink A., Jansen B.J., and Ozmultu H.C. Use of query reformulation and relevance feedback by Excite users. Internet Res.: Electron. Networking Appl. and Policy, 10(4):317–328, 2000.
4. White R.W., Ruthven I., and Jose J.M. A study of factors affecting the utility of implicit relevance feedback. In Proc. 31st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 2005, pp. 35–42.

Relevance Propagation

▶ Propagation-Based Structured Text Retrieval

Remote Method Invocation

▶ RMI

Removing Overlap

▶ Processing Overlaps

Rendezvous

▶ Workflow Join

Replica and Concurrency Control

▶ Replicated Database Concurrency Control

Replica Consistency

► Consistency Models For Replicated Data

Replica Control

Ricardo Jimenez-Peris, Marta Patiño-Martinez
Universidad Polytecnica de Madrid, Madrid, Spain

Synonyms

Database replication; Data replication protocols; Cluster database replication

Definition

In database replication, each data item has several physical copies, also called replicas, that are distributed over different nodes (sites). In case of full replication, each data item has a copy on each site. In this case, the term replica can also refer to a node hosting a copy of the entire database. *Replica control* is in charge of translating the read and write operations that clients submit on the logical data items into operations on the physical data copies. The goal is to keep a consistent state among all the replicas and to provide a consistent view of the data to the client. Replica control extends concurrency control in order to coordinate the execution of concurrent transactions at different replicas.

Historical Background

Replica control in databases has been studied since the 1980's. Early approaches mostly explored distributed locking and had their main focus on providing high availability and enforcing strong consistency in the advent of site failures and network partitions [3]. In particular, the correctness criterion 1-copy-serializability requires the execution in a replicated system to be equivalent to a serial execution over a non-replicated database.

In the 1990's a seminal paper from Gray et al. [7] criticized existing database replica control protocols providing analytical evidence of the lack of scalability due to the high deadlock probabilities when the number of replicas increases. This paper raised a new wave of research around data replication searching for scalable approaches. Some of these approaches explore weaker levels of consistency (e.g., [2,6,15]). Others aim at maintaining strong levels of consistency while still offering good scalability (e.g., [1,8,11]), sometimes relying on powerful abstractions such as group communication. Snapshot isolation has been explored as a new isolation level [5,9,14] to allow for more concurrency in the replicated system. A considerable effort has been undertaken to optimize those parts of replication control that have a large impact on the performance. Both kernel based replication solutions (e.g., [8,11] as middleware-based replication tools (e.g., [1,9,14] have been developed. Different solutions have been developed for LAN and WAN environments.

All these efforts have led to a wide range of replica control protocols. Which one to choose for a specific scenario depends on the environment and the application requirements. A trade-off between correctness, performance, generality, and potential of scalability is nearly always unavoidable.

Foundations

Replica control algorithms and their implementations can be categorized by a wide range of parameters.

Architecture

One aspect of replica control is *where* the protocol is implemented. It can be implemented within the database what is known as *kernel-based* or *white box* approach (e.g., Postgres-R [8] and the database state machine [11]). A client connects to any database replica which then coordinates with the other replicas. Typically, replica control is tightly coupled with the concurrency control mechanism of the database system.

Alternatively, replica control can be implemented outside the database as a *middleware* layer (e.g., [1,9,10,14,15]). Clients connect to the middleware that appears as a database system. The middleware then controls the execution and directs the read and write operations to the individual database replicas. This can be instrumented in two ways: (i) A *black-box approach* uses standard database systems to store the database replicas [1,14,15]; (ii) A *gray-box* approach expects the database system to export some minimal functionality that can be used by the middleware for a more efficient implementation of replica control, e.g., providing the tuples that a transaction has updated so far in form of a writeset [10]. A middleware-based approach typically has its own concurrency control mechanism which might partially depend on the concurrency control of the underlying database systems. There might be a single middleware component (centralized approach), or the middleware might be

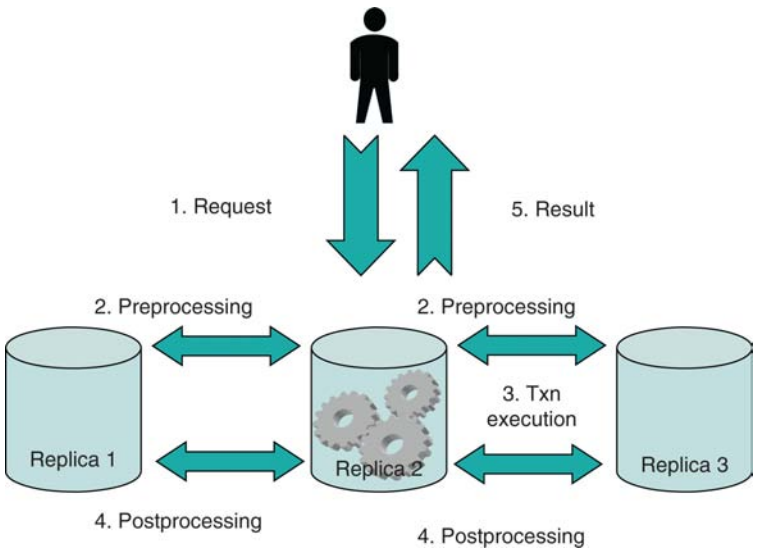
replicated itself. For example, the middleware could have a backup replica for fault-tolerance. Other approaches have one middleware instance per database replica, and both together build a replication unit.

Replica Control Phases

A replica control protocol has to control the execution of transactions during regular operation (no failure), while failures occur, and during recovery (failed sites with stale data replicas rejoin the system or new sites are added to the system). As shown in Fig. 1, the execution of a transaction during regular operation may have a number of phases (adjusted from [12]):

1. *Client connection.* The client interacts with the database through a client proxy (such as a JDBC or an ODBC driver, or the database client API). The client connects to the replicated database by invoking the client proxy connection method. In the middleware approach, the client connects to the middleware (or one of the middleware instances if they are replicated). In the kernel-based approach, it connects to one of the database replicas. Connecting to a replicated database may require some mechanism to enable *replica discovery* in a transparent way (such as a well-known registry or IP multicast). With this, the client can connect to the database independently of which replicas are currently available. If transparency is required, i.e., the client application should not be aware of replication, then replica discovery can be hidden in the client proxy.
2. *Request submission.* The client submits a request via the client proxy that forwards it to the middleware or database replica to which it is connected.
3. *Pre-processing coordination.* Some protocols require sending requests to all replicas. As a result, the replica to which the client is connected might forward the request to one, some or all replicas. This phase may also be used for other purposes such as distributed concurrency control or load-balancing.
4. *Request processing.* The request is processed by one or more replicas.
5. *Post-processing coordination.* Once a request is processed some protocols perform concurrency control tasks, propagate changes, aggregate results from a quorum, or guarantee atomicity by executing a two-phase commit protocol or another termination protocol.
6. *Result return.* The result of the request is sent back to the client proxy that returns it to the client.

A request can be either an entire transaction (consisting of one or multiple read and write operations), or an operation within a transaction. In the latter case, phases 2–6 may be repeated for each operation and for the final commit of a transaction (as in distributed locking [3]). Moreover, the order of the last two phases could be reversed, i.e., first a result is sent to the client and after that, some coordination takes place.



Replica Control. Figure 1. Replica control phases.

Replica control protocols are implemented in the pre-processing phase, in the post-processing phase or in both of them. five phases are independent of the architecture of the replicated database (replica control is implemented as a middleware layer, either centralized or distributed, or is a kernel based implementation).

Mapping Approaches

One of the main tasks of replica control is to map read and write operations on logical data items to operations on the physical data copies such that replicas eventually converge to the same value and reads see consistent data. With ROWA (read-one-write-all) protocols, read-only operations are processed at a single replica, while write operations are executed at all replicas. The extension to ROWAA (read-one-write-all-available) protocols aims at handling site failures. However, in case of network partitions ROWAA might result in two partitions executing transactions. In contrast, quorum-based replica control can handle both site and network failures. Both read and write operations need to access a quorum of replicas. Any two write quorums, and each read and write quorum have to overlap in at least one replica in order to guarantee data consistency and consistent read operations.

Correctness Criteria

One of the main aspects of replica control is the correctness criterion to be supported. Several, relatively strong correctness criteria extend the notion of isolation from a non-replicated database to a replicated setting. *1-copy correctness* states that the replicated data should appear as one logical non-replicated database. Depending on the isolation level used in the non-replicated execution different correctness criteria can be defined. In a non-replicated database, *serializability* guarantees that the concurrent execution of transactions is equivalent to a serial execution. *1-copy-serializability* [3] (1CS) extends this notion and guarantees that the concurrent execution of transactions over the replicated database is equivalent to a serial execution over a non-replicated database. Non-replicated database systems usually offer more relaxed forms of isolation, such as the ANSI isolation levels or snapshot isolation. 1-copy-snapshot-isolation has been defined as a correctness criterion for a replicated database (e.g., [9], and several replica control protocols consider snapshot isolation (e.g., [5,9,14]).

Weak consistency models do not require 1-copy correctness. Depending on the consistency level, clients might read replicas that are stale, that is, have not yet applied the latest updates, or copies might even diverge if different replicas are allowed to concurrently apply conflicting updates [16]. The degree of staleness or divergence might be bound.

Concurrency Control

In cases where *1-copy* correctness is required, distributed concurrency control is needed to enforce the correctness criteria by restricting the execution of concurrent conflicting transactions. That is, many replica control protocols extend concurrency control to a replicated system.

Concurrency control protocols can be either optimistic or pessimistic. A *pessimistic* approach restricts concurrency to enforce consistency across replicas. The easiest way consists in executing all update transactions sequentially in the same order at all sites, what would provide 1CS trivially. Other protocols increase concurrency by exploiting knowledge about the data that will be accessed. With this information, transactions accessing disjoint data sets can be executed in parallel while those that potentially access common data are executed sequentially. The granularity of data can be at different levels, such as tables, tuples or conflict classes (i.e., data partitions). Pessimistic replica control protocols are implemented in the pre-processing phase. *Optimistic* approaches, in contrast, execute potentially conflicting transactions concurrently. Only when the transaction has completed execution, a validation phase (also known as certification) takes place. It checks whether the transaction being validated conflicts with concurrent transactions. If there is a conflict, some transaction must be aborted. A standard mechanism to guarantee serializability is to abort the validating transaction if the set of data items it read during execution overlaps with the set of data items written by a concurrent transaction that already validated. With snapshot isolation, a transaction fails validation if it wrote some data item that a concurrent, already validated transaction also wrote.

Processing Update Transactions

Another important feature of replica control is how update transactions are processed. With *symmetric update processing* each update transaction is fully executed at all replicas. In contrast, *asymmetric update*

processing executes update transactions at one site (or subset of sites) and then, the resulting changes (known as *writeset*) are propagated to the rest of the replicas. Some protocols [1] lie in between, being symmetric at the statement level, but asymmetric at the transaction level. That is, if an update transaction contains both write and read statements, the read statements are executed at one site while write statements are executed at all sites. Asymmetric update processing has typically much less overhead than symmetric update processing and thus, allows for better scalability in update-intensive environments [8].

Timepoint of Synchronization

As described earlier, a client typically connects to one replica and submits its transactions to this replica. An important question is *when* this replica coordinates with other replicas to guarantee data consistency. In *eager* (aka synchronous) protocols, coordination takes place before the transaction commits locally. Typically, this means that the replica sends the changes (asymmetric processing) or the operation request (symmetric processing) and the concurrency control component decides on a serialization order for this transaction before it commits. In contrast, with *lazy replication* (aka asynchronous), updates are asynchronously propagated after the transaction commits. Lazy replication usually applies asymmetric processing. The propagation can be done immediately after the commit, periodically, or be triggered by some weak consistency criteria such as freshness. With lazy replication, transaction execution has usually no preprocessing coordination phase, and phases five and six are switched. This means, the results are first returned to the client, and then, the post-processing phase is run. Eager replication usually results in longer client response times since communication is involved but can more easily provide strong consistency.

Who Executes Transactions

Primary copy replication requires that all update transactions are executed at a given site (the primary) (e.g., [5,14,15]). The primary propagates the changes to the other replicas (secondary replicas). Secondaries are only allowed to execute read-only transactions themselves. In order to be able to forward read-only transactions to secondaries and update transactions to the primary, the system must be aware at the start time of a transaction whether it is read-only or not (e.g., through a tagging mechanism). If update transactions can be executed at any site, the replica control protocol follows an *update everywhere* approach (also referred to as *update anywhere*). An alternative to primary copy replication and update everywhere is based on partitioning the database items such that each partition has a primary, but different partitions may have different primaries (e.g., [4,10]). This avoids that the primary becomes a bottleneck under write-intensive workloads. However, it is only suitable for applications where the database can be partitioned such that each transaction only accesses data of one partition.

Degree of Replication

A further dimension is the *degree of replication*. In *full replication* every data item is replicated at each site. At the other extreme, in a *distributed database* each data item is stored at only one site and there is no replication. In partial replication each data item is replicated at a subset of nodes.

Coordination Steps

Another aspect considers the number of coordination steps, and thus, the number of message rounds, per transactions. Some protocols use a constant number of message rounds, while others require a linear number of message rounds, depending on the number of (write) operations within the transaction (e.g., in distributed locking). In the former case, the pre-processing and/or the post-processing phases are executed once for each transaction. In the latter case, these phases are executed per (write) operation.

Furthermore, some replica control protocols require a coordination protocol among the replicas in order to decide the outcome of a transaction (*voting termination*), similar to a distributed commit protocol. In others, each replica decides by itself deterministically about the outcome of a transaction (*non-voting termination*).

In early eager approaches, atomicity was achieved by running a commit protocol, such as two-phase-commit, at the end of transaction. This was not only time-consuming by itself but also required that all sites had completely executed the transaction before the transaction was committed at any site. In more recent approaches, atomicity is often achieved by other means such as reliable multicast that provides the required failure atomicity. Even in eager approaches where the participating sites agree on a serialization order before

the transaction commits at any site, this allows the user to receive the commit outcome before the transaction is actually executed at all replicas.

Restrictions

Another point to consider when analyzing replica control protocols are the possible constraints they set on the kind of transactions that are supported. Some protocols only allow single statement transactions (known as auto-commit mode in JDBC). Other protocols allow several statements within a transaction, but they have to be known at the beginning of the transaction. This is typically implemented using stored procedures (or prepared statements in JDBC) [1,10]. The more general protocols do not have any restriction on the number of statements a transaction contains [8,9,14].

Other Aspects

Replica control also needs to work correctly in the case of failures and when new or repaired replicas are added to the system. Replica control can also be affected by the implementation of self-* properties or autonomic behavior.

Another crucial feature of replica control is the environment for which it is designed. Many approaches target local area networks (LAN) where the bandwidth is high and partitions are rare. Some approaches have explored how to attain acceptable response times in wide area networks (WANs). Other kinds of networks have also been explored such as mobile networks and peer-to-peer networks. There are specific entries for all them.

Key Applications

Replica control is always needed if replicated data is updated. In particular, in database replication, where updates occur in the context of transactions, replica control is also needed to guarantee transactional correctness.

Future Directions

An important future direction is to combine replica control with modern multi-tier and service-oriented architectures. Providing high availability and scalability for a multi-tier architecture is a non-trivial task. It implies replicating all the tiers to avoid single points of failure and performance bottlenecks. In this setting, guaranteeing consistency is very challenging, especially in the face of failures. Some initial work has already been started in this direction [13] and future developments are expected along this line.

Cross-references

- ▶ Autonomous Replication
- ▶ Online Recovery in Parallel Database Systems
- ▶ Optimistic Replication and Resolution
- ▶ Replication Based on Group Communication
- ▶ Replication for High Availability
- ▶ Replication for Scalability
- ▶ Strong Consistency Models for Replicated Data
- ▶ Traditional Concurrency Control for Replicated Databases
- ▶ WAN Data Replication

Recommended Reading

1. Amza C., Cox A.L., and Zwaenepoel W . Distributed versioning: consistent replication for scaling back-end databases of dynamic content web sites. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2003, pp. 282–304.
2. Bernstein P.A., Fekete A., Guo H., Ramakrishnan R., and Tamma P. Relaxed-currency serializability for middle-tier caching and replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 599–610.
3. Bernstein P.A, Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, 1987.
4. Breitbart Y., Komondoor R., Rastogi R., Seshadri S., and Silberschatz A. Update propagation protocols for replicated databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999, pp. 97–108.
5. Daudjee K. and Salem K. Lazy Database Replication with Snapshot Isolation. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 715–726.
6. Gançarski S., Naacke H., Pacitti E., and Valduriez P. The leganet system: Freshness-aware transaction routing in a database cluster. Inf. Syst., 32(2):320–343, 2007.
7. Gray J., Helland P., O'Neil P., and Shasha D. The Dangers of Replication and a Solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
8. Kemme B. and Alonso G. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000, pp. 134–143.
9. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based data replication providing snapshot isolation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 419–430.
10. Patiño-Martínez M., Jiménez-Peris R., Kemme B., Alonso G. MIDDLE-R: Consistent database replication at the middleware level. ACM Trans. Computer Syst., 23(4): 375–423, 2005.

11. Pedone F., Guerraoui R., and Schiper A. The Database State Machine Approach. Distrib. Parall. Databases, 14(1), 2003.
12. Pedone F., Wiesmann M., Schiper A., Kemme B., and Alonso G. Understanding replication in databases and distributed systems. In Proc. 20th Int. Conf. on Distributed Computing Systems, 2000, pp. 464–474.
13. Perez-Sorrosal F., Patiño-Martínez M., Jiménez-Peris R., and Kemme R. Consistent and scalable Cache replication for multi-tier J2EE applications. In Proc. ACM/IFIP/USENIX 8th Int. Middleware Conf., 2007, pp. 328–347.
14. Plattner C. and Alonso G. Ganymed: scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004, pp. 155–174.
15. Röhm U., Böhm K., Schek H.-J., and Schuldt H. FAS – A Freshness-Sensitive Coordination Middleware for a Cluster of OLAP components. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 754–765.
16. Saito Y. and Shapiro M. Optimistic replication. ACM Computer Surveys 37(1):42–81, 2005.

Replica Freshness

Alan Fekete
University of Sydney, Sydney, NSW, Australia

Synonyms

Divergence control; Freshness control; Incoherency bounds

Definition

In a distributed system, information is often *replicated* with copies of the same data stored on several sites. Ideally, all copies would be kept identical, but doing this imposes a performance penalty. Many system designs allow replicas to lag behind the latest value. For some applications, it is acceptable to use out-of-date copies, provided they are not too far from the true, current value. *Freshness* refers to a measure of the difference between a replica and the current value.

Historical Background

The tradeoff between consistency and performance or availability is an old theme in distributed computing. In the database community, many researchers worked on ideas connected with explicitly allowing some discrepancy between replicas during the late 1980s and early 1990s. Early papers identified many of the diverse freshness measures discussed here, from groups at Princeton, Bellcore and Stanford [1,10,11]. A mixed model that integrated freshness limits of several kinds was Pu's epsilon-serializability [6,7]. TACT is a more recent mixed model, introduced by Yu and Vahdat [12]. Much of the research since the 1990s focused on optimization decisions to improve the performance of a system with slightly stale replicas. Many different system assumptions and metrics to optimize have been considered. Among influential papers are [4,5,9]. Research continues on defining different models which bound the staleness of replicas. Unlike earlier work, the focus has recently been on setting bounds which relate to the clients' view of divergence, rather than to the underlying state of the replicas. Röhm et al. designed a system with transaction-level client-defined staleness limits for read-only transactions, within the framework of 1-copy serializability [8]. Guo et al. suggested SQL extensions to express query-specific limits on the perceived staleness and inter-object drift [3], and later developed a theory to express these constraints even when stale reads are allowed within update transactions [2].

Foundations

Consider a distributed system, where information is stored at multiple sites, connected by a communications network. If several sites all store values that are intended to represent the same information in the real world, one can call these copies or *replicas*; it is usual to write x^A to represent the replica at site A of the logical data item x. At any instant, the true or current value of the logical item is the value assigned to x in the most recent non-aborted update of x. The ideal, of course, has this value stored in every replica, always (or at least, at any instant when a read operation can occur). This is possible using traditional eager replication (see entry on Traditional Concurrency Control for Replicated Databases). However, many systems prefer to propagate updates lazily, and to allow replicas or cached copies that are not completely up-to-date, as described in the entry on Optimistic Replication and Resolution. While there are many applications that can tolerate data that is not current, there are usually limits on the application's tolerance for inaccuracy. Thus many system designs allow for a bound to be placed on how far a replica can diverge from the true value; such a bound expresses a constraint on the *freshness* of the replica. This entry describes some of the main ideas that have been proposed to quantify the freshness of replicas.

The focus here is on the properties that define what is allowed, rather than on implementation details that

control how these bounds are enforced. For example, some system designs have a single master replica, which always has the true value, while in others the most up-to-date information is sometimes found in one copy and sometimes in another. However, the definitions can be stated without concern for this issue. Another axis of variation is whether the bounds have to apply always, or only when a replica is used in a query. Some papers give a numeric value for freshness or precision (which should be high), others measure staleness or imprecision. This paper uses measures where a low value is better, so zero means that the replica is completely up-to-date.

Value-Based Divergence

Suppose the logical data item comes from a numeric domain such as real numbers. In that case, one can just use the metric on the values themselves. For example, if the true current value is 10.5 and a replica has the value 9.2, then the divergence of that replica is measured as 1.3. In some proposals, the interest is in the value-based divergence between two replicas, even if neither has the current true value. If each replica is within δ of the true value, the difference between any pair of replicas is bounded above by 2δ.

Delay-Based Staleness

For some applications, a useful measure of tolerance for imprecise data is to quantify how recently the data was correct. For example, when a person moves house, the post-office will often redirect mail that had the former address, for a short period. Thus staleness can be measured by how long the replica has to wait before learning of an update that has occurred. For example, suppose the true value was 9.2 until time 100, and which time the value was updated to 10.5; if at time 103 a replica still contains 9.2, one says it is stale by 3 time units. Where the values are representing a real-world quantity, and the quantity can't change by more than v units each time interval, then a bound of δ on the delay implies a bound of $v\delta$ on the value divergence.

Many system designs can't determine, at a replica, how stale it is. Instead they keep a timestamp with each value, indicating when that update instruction was first applied. If they bound the difference between the timestamp and the current time, they also bound the staleness. Suppose one wants to keep delay below 5 time units; one can say that this holds at time 103, if the replica contains a value whose timestamp is 98 or greater. However, the argument does not work in reverse: even at time 103, a replica with timestamp 93 might be stale by less than $103 - 93 = 10$, depending on when the next update occurred after time 93. One says that a value has a valid period, which is a half-closed interval from the time at which the value first appeared in an update, until just before the next update occurred which changed the value.

Measures of Missed Updates

In many system designs, only a subset of updates are propagated to the replica (in order to save bandwidth). This motivates a definition where one measures how many updates occurred on the logical item, without being recorded (yet) at the replica. For example, suppose the logical item is updated from 9.2 to 10.5 at time 100, and then to 10.1 at time 102, and then to 9.6 at time 104. A replica with value 9.2 at time 103 has missed 2 updates that have occurred by that time. This definition can be related to a delay-based measure if the updates come periodically, as is common for sensor readings.

Inter-Object Consistency Drift

When a query reads several logical objects, one might bound the drift between the versions seen in the two reads. For example, if a query examines copies of the temperature and humidity from a sensor, the user might care that the two measurements were taken at almost the same time, because the humidity affects the accuracy of the thermometer. Suppose a replica x^A contains a value whose valid period is the interval $[T,U)$ and y^B contains a value valid in the interval $[V, W)$. Measure the drift between the replicas, as the smallest separation between the valid intervals; this is zero if $[T,U) \cap [V,W) \neq \emptyset$, and otherwise it is $min(|V - U|, |T - W|)$. A drift bound of zero is called a "snapshot" condition, as all the data examined must come from a common state of the database.

Transaction Semantics

Often, each update and each query is a separate operation. However, several updates or several queries can be placed in a single transaction. In 1-copy serializability (q.v.), all the operations in a transaction have to appear to take place together, but a transaction with reads may be serialized in the past, and so one might desire a common maximum delay for all the reads done in that transaction. A different approach is taken in epsilon-serializability. Here one ascribes a numeric measure to each of the various situations of divergence

between the value read and the correct value, and adds up the divergence measure seen in each read, there is then a bound on the total divergence accumulated during the reads in a given query transaction.

Mixed Measures

Each of the definitions of freshness seems to work for some applications but not for others. Thus the TACT model has separate bounds of each measure which must be applied to all the logical data items within a user-specified grouping of items. For example, a particular logical item might require that whenever it is read, it is separated from the true value by no more than 1.5 in numeric value, it is stale by no more than 10 in delay, and it must be missing no more than 3 updates done at the master.

Key Applications

The commercial database platforms generally offer best-effort update propagation, rather than giving applications guaranteed bounds on freshness. Thus the properties described here are usually found in research prototypes or special-purpose data management, for example in cache management for web content or sensor data.

Future Directions

The ideas of replica freshness reappear in new domains, where applications might be able to tolerate some imprecision and where there is a high cost to keeping data up-to-date. For example, somewhat stale data might be used in a sensor network, or delivered in dynamic web page content. Many of these ideas are being generalized, in a broad research agenda that deals with uncertain or imprecise data.

Cross-references

- ▶ Data Acquisition
- ▶ Data Replication
- ▶ Real-Time Transaction Processing
- ▶ Sensor Networks
- ▶ Uncertainty Management in Scientific Database Systems

Recommended Reading

1. Alonso R., Barbará D., and Garcia-Molina H. Data caching issues in an information retrieval system. ACM Trans. Database Syst., 15(3):359–384, 1990.
2. Bernstein P.A., Fekete A., Guo H., Ramakrishnan R., and Tamma P. Relaxed-currency serializability for middle-tier caching and replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 599–610.
3. Guo H., Larson P.-Å., Ramakrishnan R., and Goldstein J. Relaxed currency and consistency: how to say "good enough" in SQL. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 815–826.
4. Olston C., Loo B.T., and Widom J. Adaptive precision setting for cached approximate values. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 355–366.
5. Pacitti E., Coulon C., Valduriez P., and Özsu M.T. Preventive replication in a database cluster. Distrib. Parall. Databases, 18(3):223–251, 2005.
6. Pu C. and Leff A. Replica control in distributed systems: as asynchronous approach. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991, pp. 377–386.
7. Ramamritham K. and Pu C. A formal characterization of epsilon serializability. IEEE Trans. Knowl. Data Eng., 7(6):997–1007, 1995.
8. Röhm U., Böhm K., Schek H.J., and Schuldt H. FAS – a freshness-sensitive coordination middleware for a cluster of OLAP components. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 754–765.
9. Shah S., Ramamritham K., and Shenoy P.J. Resilient and coherence preserving dissemination of dynamic data using cooperating peers. IEEE Trans. Knowl. Data Eng., 16(7): 799–812, 2004.
10. Sheth A.P. and Rusinkiewicz M. Management of interdependent data: specifying dependency and consistency requirements. In Proc. Workshop on the Management of Replicated Data, 1990, pp. 133–136.
11. Wiederhold G. and Qian X. Consistency control of replicated data in federated databases. In Proc. Workshop on the Management of Replicated Data, 1990, pp. 130–132.
12. Yu H. and Vahdat A. Design and evaluation of a conit-based continuous consistency model for replicated services. ACM Trans. Comput. Syst., 20(3):239–282, 2002.

Replicated Database Concurrency Control

Bettina Kemme
McGill University, Montreal, QC, Canada

Synonyms

Replica and concurrency control

Definition

In a replicated database, there exist several database servers each of them maintaining a (partial) copy of the database. Thus, each logical data item of the database has several physical copies or replicas. As transactions submit their read and write operations

on the logical data items, the *replica control* component of the replicated system translates them into operations on the physical data copies. The *concurrency control component* of the replicated system controls the execution order of these operations such that the global execution obeys the desired correctness criteria. For a tightly coupled system with strong consistency requirements, 1-copy-serializability is the standard correctness criterion, requiring that the concurrent execution of transactions on the replicated data has to be equivalent to the serial execution of these transactions over a logical copy of the database. Thus, a global concurrency control strategy is needed. For loosely coupled systems with weak consistency requirements, each database server uses its local concurrency control mechanism. Inconsistencies are only resolved later via reconciliation techniques.

Key Points

Global concurrency control strategies for replicated databases can be centralized or distributed. In a centralized solution, there is a single or main concurrency control module which could be installed on one of the database servers or in an independent component (e.g., a middleware layer between clients and the individual database servers). This single scheduler decides on the global execution order. Using this architecture, standard concurrency control methods found in non-replicated database systems can be easily extended to the replicated environment. However, the single module becomes a single point of failure, and might lead to considerable message overhead. In a distributed solution, each database server has its own concurrency control component, and the different components have to work in a coordinated fashion in order to guarantee the correct global execution order.

The standard strict two-phase-locking (2PL) protocol can be extended very easily to a replicated system. In a centralized solution, there is no real difference to a non-replicated database. Before an operation on a data item is issued, a lock for this data item has to be acquired and all locks are released at commit time. In a distributed solution, each local lock manager performs strict 2PL. The difference is that locks are now set on the physical copies and not on the logical data item. As many replica control algorithms follow the read-one-copy/write-all-copies strategy it means that a write operation on a data item will lead to write locks on all database servers holding a copy of the data item. With this, a distributed deadlock might occur that involves only a single data item.

Optimistic concurrency control mechanisms are attractive in a replicated database system since they allow a transaction to be executed first locally at only one database server. Only at commit time, the modified data items are propagated to the other replicas and a validation phase checks whether a proper global execution order can be found. This keeps the communication overhead low.

In general, communication is an important issue in a replicated database. An option is to take advantage of advanced communication primitives, e.g., multicast protocols that provide delivery guarantees and a global total order of all messages. As all available database servers receive all messages in the same total order, this order can be used as a guideline to determine at each site locally the very same global serialization order.

Cross-references

- ▶ Concurrency Control – Traditional Approaches
- ▶ Data Replication
- ▶ One-Copy-Serializability
- ▶ Replica Control
- ▶ Replication Based on Group Communication
- ▶ Serializability
- ▶ Traditional Concurrency Control for Replicated Databases

Recommended Reading

1. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency control and recovery in database systems. Addison Wesley, Reading, MA, 1987.
2. Carey M.J. and Livny M. Conflict detection tradeoffs for replicated data. ACM Trans. Database Syst., 16(4):703–746, 1991.
3. Wiesmann M. and Schiper A. Comparison of database replication techniques based on total order broadcast. IEEE Trans. Knowl. Data Eng., 17(4):551–566, 2005.

Replication

Kenichi Wada
Hitachi Limited, Tokyo, Japan

Synonyms

Data Replication; Duplication

Definition

Replication (or data replication) is the process to make copy of a collection of data. Copied data made by replication is referred as replica (or data replica).

Replication can be categorized by data type to be copied. Most DBMSs support database replication for high availability or parallel transaction processing. Some filesystems support filesystem replication for high availability or workload distribution. Filesystem replication is commonly used in distributed filesystems. Volume replication can be processed by Logical Volume Manager, Filesystem or Storage system.

Most replication techniques try to keep replicas consistent and updated so that in case of system failures, the replica can be used to recover from the failure with minimal data loss.

Key Points

During data replication, original data is usually copied on an ongoing basis. The goal of this data replication is to provide high availability. There are two types of data replication in terms of data copy method: synchronous and asynchronous. Synchronous replication guarantees, "zero data loss" by means of atomic write operations (i.e., a write either completes on both sides or not at all). A write is not complete until there is acknowledgment for both original and replica. Most applications wait for a write transaction to complete before proceeding with further work, so synchronous replication may lower overall performance considerably.

In asynchronous replication, a write is complete as soon as the original is updated. The replica is usually updated after a small lag. This decreases write costs, but "zero data loss" is not guaranteed.

Data replication can be categorized by the layer of doing it. In case of Server-based replication, some entity (e.g., DBMS, Filesystem, Logical Volume Manager, and Application) in a server performs replication. Most Database replication and filesystem replication falls into this category.

Appliance-based replication is another category. Some appliance in a network connecting servers and storage systems makes replication. Most storage virtualization appliances have volume replication functions. File virtualization appliances may provide filesystem replication.

Storage-based replication is done by storage system. Block storage can provide volume replication and file storage such as NAS can provide filesystem replication.

Cross-references

▶ Backup and Restore
▶ Data Replication

Replication Based on Group Communication

Fernando Pedone
University of Lugano, Lugano, Switzerland

Definition

Database replication based on group communication encompasses protocols that implement database replication using the primitives available in group communication systems. Most commonly, these replication protocols ensure strong data consistency (e.g., serializability, snapshot isolation), which translates into ordering requirements on the data operations. Protocols not based on group communication primitives should implement this ordering from scratch, typically in some ad hoc manner. Implementing ordering guarantees is not an easy task, notably in the presence of failures since it is usually difficult to determine the state a failed site was in when the crash occurred. Group communication primitives offer message reliability and ordering properties that simplify the design of replication protocols. As a consequence, database replication protocols based on group communication are usually highly modular.

Historical Background

Group communication was introduced in the 1980s. The Isis system pioneered work on group communication primitives that provide strong guarantees in the presence of failures [4], a concept that builds on the earlier abstraction of process groups to support one-to-many communication. Database replication based on group communication was introduced in the 1990s.

To understand how group communication influenced database replication, the first step is to provide a brief account of how group communication is related to replication in distributed systems in general. Many fault-tolerant distributed systems aiming at strong consistency are implemented according to the state-machine replication approach [10,15], sometimes called active replication. (Notice that while [15] extends state-machine replication to the context of

failures, the concept was introduced in [10] for environments in which failures could not occur.) Essentially, the state-machine replication approach can be decomposed into two requirements, which orchestrate the dissemination of operations to replicas:

Agreement. Every non-faulty replica receives every operation.
Order. Every non-faulty replica processes the operations it receives in the same relative order.

Assuming that each replica can be made to behave as a state machine, when provided with the same operations in the same order, all replicas will produce the same results. The problem is then how to ensure the agreement and order properties. It turns out that group communication systems provide primitives that guarantee these properties. Thus, it comes perhaps as no surprise that group communication has become a basic building block for implementing state-machine replication, and fault-tolerant distributed systems.

Two observations from the mid-1990s help understand how group communication came into play in database replication. First, it was pointed out that transactions in replicated databases share common properties with group communication primitives [14]. For example, transaction isolation translates into ordering requirements among requests, a property typically available in group communication systems. Likewise, transaction atomicity is usually referred to as agreement in group communication systems. Second, it was shown that mechanisms typically used to provide strong consistency in distributed databases are inappropriate in replicated scenarios. For example, distributed two-phase locking has an expected deadlock rate that grows with the third power of the number of replicas [7]. The underlying argument from these results follows from the fact that out of order requests may get entangled, increasing the chances of deadlocks, a shortcoming that group communication primitives can easily solve.

As a consequence, researchers started looking at database replication protocols based on primitives that provide guarantees similar to those found in transactional systems.

Foundations

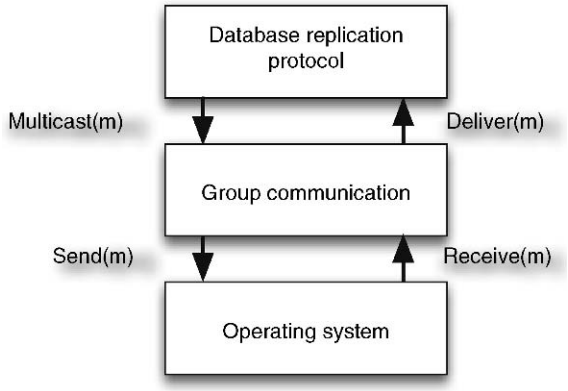
How can group communication help design database replication protocols? To answer this question group communication primitives are discussed and then a framework is presented to reason about the usefulness of these primitives in database replication. A recurrent approach to replication based on group communication is the deferred update model, which is presented next. In this discussion, unless stated otherwise, servers are assumed to have a full copy of the database.

Group Communication Primitives

The abstraction of groups (of servers) acts as a logical addressing mechanism in a replicated system, allowing the degree of replication and the identity of the individual servers to be ignored. For example, a group can abstractly represent the servers that replicate the database. Moreover, group communication primitives provide one-to-many communication with various powerful semantics, and hide much of the complexity of maintaining the consistency of the system and handling failures of replicated servers.

Group communication is usually characterized by two primitives: multicast(m, g) allows a message m to be forwarded to the members of group g; the message is passed to the application by means of the deliver (m) primitive. Multicast and deliver are implemented on top of simpler communication primitives (i.e., providing weaker guarantees). A multicast message is delivered to the application after the members of the group the message is addressed to have interacted, possibly exchanging point-to-point messages, to implement the properties ensured by the group communication (see Fig. 1).

Group communication primitives typically provide *message atomicity*: if a server in g delivers m, then every



Replication Based on Group Communication. Figure 1. Group communication primitives.

non-faulty server in g should also deliver m. Notice that if a server is down when a message m is multicast, it will not be able to deliver m (i.e., the server is faulty). If the server later recovers, it should deliver all messages it has missed. This is done using a state transfer mechanism, by which a recovering server receives the missed state from operational severs before it resumes its execution. State transfer is transparent to the application and implemented by the group communication system.

Another property typically ensured by group communication primitives concerns *message ordering*. Two such properties are *first-in-first-out (FIFO)* and *total order*. FIFO ensures that messages multicast by the same sender are delivered in the order they are multicast, that is, if a server executes multicast(m, g) followed by multicast(m', g), all non-faulty servers in g will deliver m first and then m'. Total order guarantees that all messages, regardless of the sender, are delivered in the same order by all non-faulty servers. If server s executes multicast(m, g) and server s' executes multicast(m', g), then m and m' should be delivered in the same order. Notice that FIFO is not a special case of total order. With total order, if a server multicasts m followed by m', the destinations can deliver m' before m, as long as they all deliver m and m' in the same order. Both properties can be combined in a *FIFO total order* property.

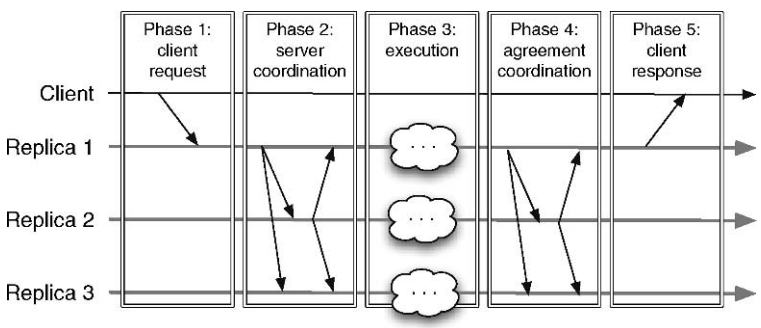
Groups can be *static* or *dynamic*. Static groups do not change their membership during the execution, although members can crash and later recover. Dynamic groups adapt the membership according to the state of their members, operational or crashed. Dynamic groups use a sequence of views, $v_0(g), v_1(g), ..., v_i(g), ...$, to manage membership. Each view $v_i(g)$ defines the composition of g at some time t, i.e., the members of the group that are operational at time t. Whenever a server in some view $v_i(g)$ is suspected to have crashed, or some server wants to join, a new view $v_{i+1}(g)$ is installed, which reflects the membership change.

Group communication primitives for dynamic groups can guarantee properties relating views to multicast messages. One such property is *sending view delivery*, according to which a message can only be delivered in the context of the view in which it was multicast. This means that if a server multicasts message m while in view $v_i(g)$, then m can only be delivered before view $v_{i+1}(g)$ is installed.

The following sections use two multicast primitives with different properties: abcast(m, g) ensures message atomicity and total order in the context of static groups; vscast(m, g) assumes dynamic groups and guarantees message atomicity, FIFO ordering, and sending view delivery properties. In both cases, messages are delivered to the application using deliver(m).

A Functional Model for Database Replication

Replication protocols can be generically decomposed into five phases, as shown in Fig. 2 [16]. Some protocols may skip some phases, while others may apply some of the phases several times during the execution of transactions. In Phase 1 the client contacts one or more database replicas to submit its requests. Phase 2 involves coordination among servers before executing requests. (Notice that message exchanges in Fig. 2 are illustrative only; different communication protocols may present different message patterns.) The actual execution of a request takes place in Phase 3; one or more servers may be involved in the execution of the operation. In Phase 4 servers agree on the result of the execution, usually to ensure atomicity, and in Phase 5 one or more servers send the result to the client. There



Replication Based on Group Communication. Figure 2. A functional model for database replication.

are basically two parts in which group communication primitives can help: during *server coordinaton* (Phase 2) and *agreement coordination* (Phase 4).

Database replication protocols can be *eager* or *lazy*, according to when update propagation takes place; and *update everywhere* or *primary copy*, according to who performs the updates. Group communication has traditionally been used by eager protocols, both in the case of update everywhere and primary copy. Some eager approaches are briefly reviewed next; a more detailed account can be found in [16].

Distributed two-phase locking (D2PL) is a traditional mechanism to coordinate servers during transaction execution. In replicated scenarios, implementing D2PL with point-to-point communication may lead to many distributed deadlocks [7]. If lock requests are propagated to servers using abcast, however, total order is ensured, and the probability of deadlocks is reduced [2]. In such a protocol, the client submits its requests to one database server (Phase 1) which abcasts them to all servers (Phase 2). Upon delivering the request, servers should make sure that conflicting locks are obtained in a consistent manner (Phase 3). Once the servers have executed the operation they send the response to the client (Phase 5). Commit and abort requests are processed like any other operation and so there is no need for agreement coordination (Phase 4). Notice that Phases 1–3,5 may be executed multiple times, once per request. If transactions can be predefined (e.g., storage procedures), then they can be abcast to all servers as a single request, and then Phases 1–3,5 are executed only once per transaction [3,11].

The deferred update approach is another form of eager update-everywhere replication mechanism based on group communication. In this case, the client selects one database server and submits all its requests to this server, without communicating with other servers (Phase 1). During the execution of the transaction, there is no synchronization among servers (i.e., no Phase 2), and only the selected server executes the transaction (Phase 3). Read-only transactions are committed locally by the selected server. Update transactions are propagated to all servers at commit time using abcast. The delivery of a terminating update transaction triggers a certification procedure used to ensure consistency. If the transaction passes the certification test its updates are committed against the database (Phase 4). Only then the reply is sent to the client (Phase 5). Several group communication-based replication protocols follow this approach, which is discussed in more detail in the next section.

Eager primary copy replication can also take advantage of group communication primitives. In this case clients interact only with the primary copy during the execution of the transactions (Phase 1). Therefore, there is no Phase 2. After the primary executes a request it sends the result to the client (Phase 3 executed by the primary only). Several requests from the client may be executed against the primary copy until commit is requested, at which point the primary communicates the new database state (e.g., redo logs) to the secondary copies (Phase 4). The secondaries apply the modifications to their local database. Communication between the primary and the secondaries is through vscast. FIFO ordering ensures that updates from the primary are received in the order they are sent. Sending view delivery guarantees correct execution in case of failure of the primary. For example, if the primary fails before all secondaries receive the updates for a certain request and another replica takes over as new primary, vscast ensures that updates sent by the new primary will be properly ordered with regard to the updates sent by the faulty primary.

Deferred Update Database Replication Protocols

Database replication protocols based on the deferred update technique, sometimes called certification-based protocols, are optimistic in that transactions execute on a single server without synchronization among servers. Abcast ensures that at termination transactions are delivered in the same order by all servers. The total order property together with a deterministic certification test guarantee that all servers agree on which transactions should be committed and which ones should be aborted.

The precise way in which the certification test is implemented and the information needed to implement it depend on the consistency criterion. Two typical consistency criteria are serializability and snapshot isolation. Serializability specifies that a concurrent execution of transactions in a replicated setting should be equivalent to a serial execution of the same transactions using a single replica. With snapshot isolation, transactions obtain at the beginning of their execution a "snapshot" of the database reflecting previously committed transactions; a transaction can commit as long as its writes do not intersect with the writes of the transactions that committed since the snapshot was taken.

The certification test may require transaction readsets and writesets. Transaction readsets and writesets refer to the data items the transaction reads and writes during its execution (e.g., the primary keys of the rows read or written). Let $RS(T)$ and $WS(T)$ denote, respectively, the readset and writeset of transaction T, and let $CC(T)$ be the set of transactions that executed concurrently with T. There are different ways to define the concurrent transactions set; typically, the set contains the transactions that committed after T started its execution, but before T itself committed. The propositions below abstractly define the conditions for committing a transaction according to serializability and snapshot isolation, respectively.

Serializability. T can commit if and only if $\forall T' \in CC(T) : RS(T) \cap WS(T') = \emptyset$
Snapshot isolation. T can commit if and only if $\forall T' \in CC(T) : WS(T) \cap WS(T') = \emptyset$

There are two general ways in which certification can be implemented in order to ensure serializability: with and without explicit information about transaction readsets. If information about data items read by the transaction is available, both the readsets and the writesets, together with the actual updates, are abcast to all servers as part of the termination of T. Upon delivering this message, each server individually evaluates the condition for serializability and determines whether T can be committed. When certifying T, each server builds $CC(T)$ based on transactions previously delivered, and in doing so, the test can be implemented deterministically, guaranteeing that all servers reach the same decision concerning committing transactions [12].

Certification without transaction readsets can be performed with an additional communication step among servers [9]. In the following, read operations are assumed to be performed before write operations. When a transaction requests to commit, only its writeset and the actual updates are abcast to all servers. After delivering the message, the server that executed the read operations tries to execute all write operations of the terminating transaction. Ongoing local transactions with read locks that conflict with the terminating transaction's write locks require special care: If the transaction has not been abcast yet, it is simply locally aborted; otherwise an abort message is abcast to all servers. Once the terminating transaction has acquired all its write locks, a commit message is abcast to all servers, which upon delivery commit the transaction.

Snapshot isolation can be implemented with a single round of messages (i.e., one abcast per transaction only) without readsets [9]. At commit time the writeset and the updates of a committing transaction are abcast to all servers. Each server executes the certification test and decides individually whether the transaction can commit or not.

Key Applications

Database replication protocols based on group communication are usually used to guarantee high availability for OLTP applications. Some of the existing protocols have been proved to provide good scalability under read-intensive workloads. Given the frequent assumption that each participating server should have a full copy of the database (i.e., full replication), scalability under write-intesive workloads is limited.

Future Directions

Replication based on group communication has been a hot topic of research in the past years. Several protocols have been proposed, a few implemented, and some made available for free download. While the main tradeoffs are known now, some problems are still open.

One open problem concerns how to structure group communication-based database replication protocols. According to whether changes in the database engine are required or not, there are two possibilities: kernel- and middleware-based architectures. Kernel-based protocols (e.g., [12,9]) take advantage of internal components of the database to increase performance in terms of throughput, scalability, and response time. For the sake of portability and heterogeneity, however, replication protocols should be independent of the underlying database management system. Even if the database internals are accessible, modifying them is usually a complex operation. As a consequence, middleware-based database replication (e.g., [6,13]) has received much attention in the last years. Such solutions can be maintained independently of the database engine, and can potentially be used in heterogeneous settings. Kernel-based protocols, on the other side, have more information about the data accessed by the transactions (e.g., readsets), which may result in more concurrency or less abort rate or both. In order to make its complexity more manageable, some efforts have tried to standardize kernel-based protocols [1].

Another problem subject to further investigation is partial replication. Full replication protocols have limited scalability under update-intensive workloads [8]. This is not a specific limitation of existing protocols, but an inherent characteristic of fully replicating the database on each server. Each new server added to a fully replicated system allows more clients to connect and submit transactions. If such transactions update the database, they will add load to every individual server. Partial replication does not suffer from the same problem since the degree of replication of each data item can be controlled. Thus, servers can be added to the system, improving the performance of data mostly read and without increasing the load of existing servers. It is not obvious how to implement partial replication in the context of group communication though and few proposals have considered it so far (e.g., [5]).

Cross-references

- ► Consistency Models for Replicated Data
- ► 1-Copy-Serializability
- ► Data Replication
- ► Replica Control

Recommended Reading

1. http://gorda.di.uminho.pt/
2. Agrawal D., Alonso G., Abbadi A.E., and Stanoi I. Exploiting atomic broadcast in replicated databases. In Proc. 3rd Int. Euro-Par Conference, 1997.
3. Amir Y. and Tutu C. From total order to database replication. In Proc. 22nd Int. Conf. on Distributed Computing Systems, 2002.
4. Birman K. The process group approach to reliable distributed computing. Commun. ACM, 36(12):37–53, 1993.
5. Camargos L., Pedone F., and Wieloch M. Sprint: a middleware for high-performance transaction processing. In Proc. Second European Conference on Systems Research, 2007.
6. Cecchet E., Marguerite J., and Zwaenepoel W. C-JDBC: flexible database clustering middleware. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004.
7. Gray J.N., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. 1996 ACM SIGMOD Int. Conf. on Management of Data, 1996.
8. Jiménez-Peris R., Patiño-Martínez M., Alonso G., and Kemme B. Are quorums an alternative for data replication? ACM Trans. Database Syst., 28(3):257–294, 2003.
9. Kemme B. and Alonso G. A new approach to developing and implementing eager database replication protocols. ACM Trans. Database Syst., 25(3), 2000.
10. Lamport L. Time, clocks, and the ordering of events in a distributed system. Commun. ACM, 21(7):558–565, 1978.
11. Patiño-Martínez M., Jiménez-Peris R., Kemme B., and Alonso G. Middle-R: Consistent database replication at the middleware level. ACM Trans. Comput. Syst., 23(4), 2005.
12. Pedone F., Guerraoui R., and Schiper A. The database state machine approach. Distrib. Parall. Databases, 14(1):71–98, 2003.
13. Salas J., Jiménez-Peris R., Patiño-Martínez M., and Kemme B. Lightweight reflection for middleware-based database replication. In IEEE Int. Symp. on Reliable Distributed Systems, 2006.
14. Schiper A. and Raynal M. From group communication to transaction in distributed systems. Commun. ACM, 39(4):84–87, 1996.
15. Schneider F.B. Implementing fault-tolerant services using the state machine spproach: A tutorial. ACM Comput. Surv., 22 (4):299–319, 1990.
16. Wiesmann M., Pedone F., Schiper A., Kemme B., and Alonso G. Understanding replication in databases and distributed systems. In Proc. 20th Int. Conf. on Distributed Computing Systems, 2000.

Replication for High Availability

Bettina Kemme
McGill University, Montreal, QC, Canada

Synonyms

Fault-tolerance; Backup mechanisms

Definition

Replication is a common mechanism to increase the availability of a data service. The idea is to have several copies of the database, each of them installed on a different site (machine). Using replication, the data remains available as long as one site is running and accessible. Fault-tolerance is related to availability. A system is considered fault-tolerant if it continues to work correctly despite the failure of individual components. Replicating data and processes over several sites, the failure of any individual site can be masked since the tasks executed by the failed site can be transferred to one of the available sites. The terms high availability and fault-tolerance are often used interchangeably. However, fault-tolerance is stronger than a high availability solution since it expects the fault-tolerant system to behave exactly as a system where components never fail. This requires to make failures transparent to clients and typically means that all data copies have to be consistent at all times. This is difficult to achieve in systems where network connectivity is not always guaranteed. There, a high availability solution might allow

users to access locally available data copies although it is possible that they do not reflect the latest updates.

Historical Background

High availability and fault-tolerance have received attention in many different communities covering high available hardware, fault-tolerant software systems and data availability. Redundancy is one of the major mechanisms to achieve the desired goal.

In the database community, looking for high availability solutions was the main reason to start research into database replication [3], while replication for scalability and performance were only explored later. The research community has mainly focused on the maintenance of transactional properties despite various kinds of failures. Apart from mere correctness, there exist many implementation alternatives that can have a large impact on the performance of the system, and current major database systems have developed a wide range of high availability solutions with different trade-offs.

Data availability has also been explored in many other data centric domains such as file systems [10, 12], mobile systems [6] or fault-tolerant processes [4].

Foundations

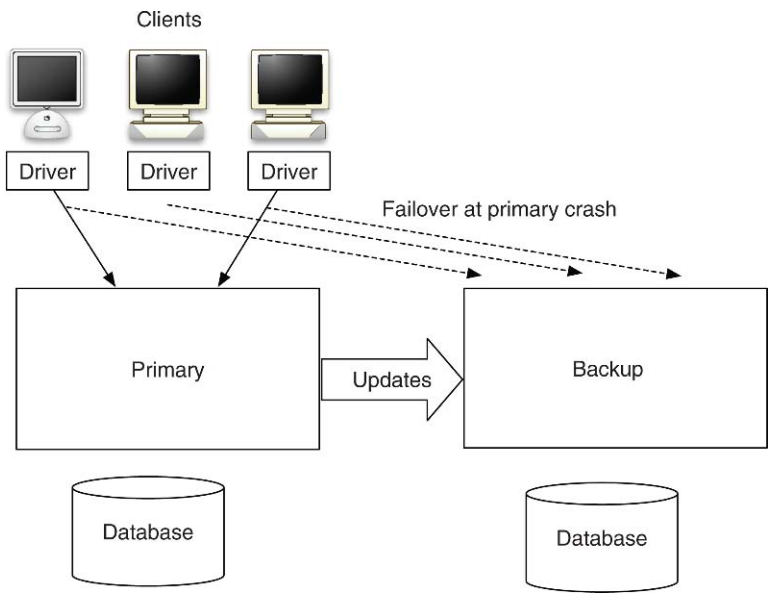
This entry focuses solely on availability for database systems. However, many of the issues and solutions discussed here are also valid outside the database domain.

Basic Fault-Tolerance Architecture

This discussion first looks at the case when a database system crashes. In this case, it does not perform any further actions and the data is no more available. Only after restart and recovery it becomes again operational. As the recovery procedure can take considerable time, a fault-tolerance solution maintains more than one copy of the database. In the following, the terms replica and site refer to a machine running a database management instance and controlling a copy of the complete database.

A fault-tolerance replication architecture has to tackle several issues. Firstly, when data is updated the data copies have to be kept consistent, resulting in additional measures, and thus *overhead during normal processing when no failures occur*. Secondly, when a replica fails, a *failover procedure* has to reconfigure the system. Thirdly, after a failure, the failed replica or a new replica have to be started and added to the system in order to handle future failures. For this, the joining replica has to receive the current state of the database from the available replicas. This third step is referred to as *replica recovery*.

As the most common fault-tolerance architecture uses primary-backup replication (Fig. 1) the following



Replication for High Availability. Figure 1. Primary-backup architecture.

discussion focuses on this architecture. There is one primary replica and one or more backup replicas in the system. All clients are connected to the primary replica which executes all transactions. In the case the primary fails, the clients have to be reconnected to one of the backups which takes over the tasks of the primary.

Execution While No Failures Occur The primary executes all transactions and decides on a serialization order (via the concurrency control module). At specific timepoints, the replication module of the primary replica propagates updates to the backup replica(s). Each backup replica applies the changes in an order that conforms to the serialization order determined by the primary.

Propagation can be done in several ways. In a *2-safe* approach, the changes performed by a transaction are propagated before the transaction commits, and an agreement protocol, such as a 2-Phase-Commit protocol, guarantees atomicity (the transaction either commits or aborts at all participating replicas). In a *1-safe* approach, the primary can commit a transaction before update propagation is completed.

1-safe propagation is faster than 2-safe propagation. However, if the primary fails after committing a transaction but before propagating its changes then the transaction is lost because the backup that becomes the new primary has no information about it. Neither atomicity nor durability of the transaction is given. Recovery of this transaction can be complex. In other context, 1-safe propagation is also often referred to as lazy, asynchronous or optimistic replication, while 2-safe propagation is often referred to as eager or synchronous replication.

There are several other design choices in regard to propagation. For instance, a message could be sent per update operation, per transaction or even per set of transactions. Furthermore, the update message could contain the physically changed records (e.g., taken from the log used for local recovery) or the update statement.

Changes can be applied at the backup at different timepoints. A backup is called *hot-standby* if it immediately applies any changes it receives to its own local database. Therefore, at the time of the failover, the backup is immediately operational and can accept client requests. However, it requires a powerful backup because of the high overhead during normal processing. A *cold-standby* defers applying the changes, e.g., to a timepoint when it is idle or when it actually becomes the new primary at failover time. This results in low overhead during normal processing. Thus, a cold-standby could be a less-expensive machine. However, failover will likely take longer and performance will be compromised until the powerful machine has completed recovery and can become again the primary.

Failover If there is more than one backup, an agreement protocol can be used to decide which of the backups takes over as new primary.

As part of the failover procedure, the clients have to be reconnected to the new primary. This reconnection is typically implemented within the driver software (e.g., JDBC driver) running on the client side (see Fig. 1). The driver is connected to the primary but knows the backup(s). When it looses connection to the primary, it connects to the backups to see who is the new primary.

Furthermore, the new primary, if is has not yet done so, has to apply all the changes of committed transactions that it received from the old primary before the crash. It aborts transactions for which it has already received some information but not all. A difficult issue is the handling of transactions that were active at the time of the failure. If the client had not yet submitted the commit request, the driver can simply throw an abort exception for this transaction before it connects to the new primary. This is correct since independently of being 1-safe or 2-safe, the new primary has neither committed nor is involved in a 2PC for this transaction. It is up to the client to resubmit the transaction (as would be the case for transactions involved in deadlocks). A different situation arises if the application program has already submitted the commit request, and the driver is waiting for the confirmation from the primary when the failure occurs. In this case, the transaction might already be committed at the backups. The driver has to ask the new primary for the outcome of the transaction and inform the client accordingly with a commit confirmation or an abort exception.

Replica Recovery Replica recovery is the task of integrating a new or failed replica into the replicated system. The recovery procedure can be performed *online* or *offline*. In offline recovery, transaction processing is halted in the system, the joining replica receives the

accurate state of the data, and then transaction processing can resume. The problem is that availability is compromised. Online recovery, in contrast, performs recovery without stopping transaction processing in the rest of the system.

Data transfer strategies. There are several possibilities to provide a recovering replica with the latest state of the database [6]. One possibility is to transfer the entire database state to the recovering replica. This approach is simple but leads to unnecessary data transfer if large parts of the data did not change during the downtime of the recovering replica. Thus, an alternative solution first determines for each data item, whether this data item was actually changed during the downtime. Only if this is the case, the current version is transferred to the recovering replica. While this might decrease the size of data to be transferred it needs additional measures during normal processing and recovery to determine the changed objects. Instead of the data items, one can also transfer the log with all updates that the recovering replica has missed. Then, the recovering replica applies these missing updates to its database. The later two approaches are only possible for failed replicas that later recover while completely new replicas need a complete state transfer.

Recovery Procedure. A failed replica or a new replica typically joins as a backup. It can become the primary once recovery has completed (e.g., if it is the most powerful machine). The joining replica can receive the state from the primary or from another backup.

Using online recovery, the state transfer takes place concurrently to transaction processing at the primary. Thus, one has to make sure that the recovering replica does not miss any transactions. That is, for a given transaction, the updates performed by this transaction at the primary are either reflected by the state transfered through recovery, or are propagated by the primary to the new backup after recovery has completed.

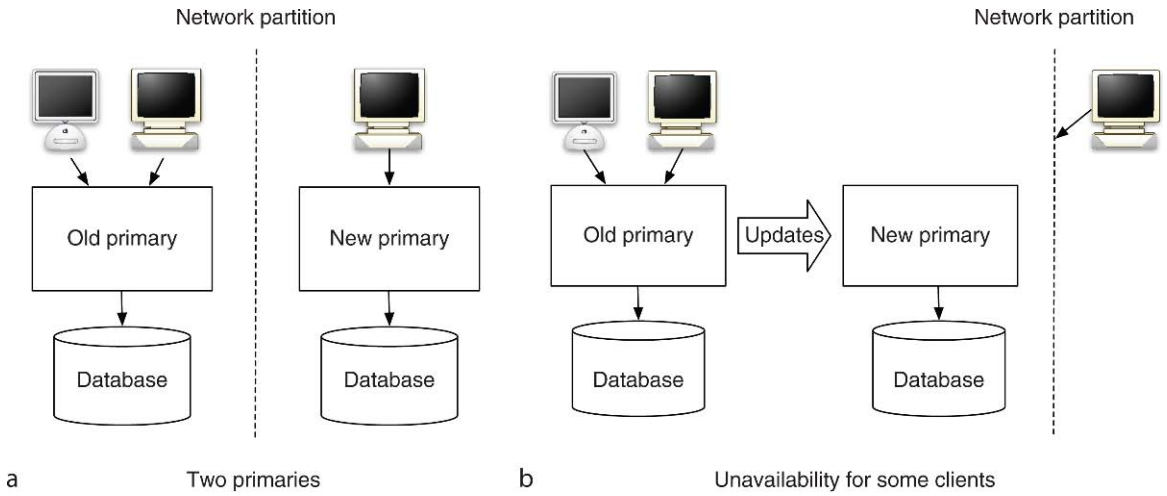
Combining Scalability and Fault-Tolerance

Many recent approaches exploit replication for both scalability and fault-tolerance by allowing transaction execution to be distributed across all available replicas (typically a LAN cluster). The protocols follow a read-one-write-all-available (ROWAA) replication strategy [2] where read operations are executed at one replica (which allows load-balancing) and write operations are executed at all replicas that are currently available (to keep copies consistent). In fact, the primary-backup approach is also a ROWAA approach but all clients are connected to the primary and transactions are only executed there. In these cluster replication solutions [9], when a replica fails, it is excluded from the system. Clients served by the replica are transferred to available replicas, and writes are only performed on the remaining available replicas. Failover is complicated by the fact of ongoing transaction execution during the failover period.

Network Partitions

Site crash is not the only failure type that can occur. A network problem might (temporarily) partition the replicas, allowing each replica to only communicate with a subset of other replicas. Network partitions occur seldom in LANs but are common in WANs or wireless networks. They are often transient and only hold for short periods of time. In general, it is impossible for a replica to determine whether another replica that is not reachable has failed, is currently not connected, or is simply slow and overloaded making it temporarily non-responsive. A correct fault-tolerance solution needs to work correctly despite this ambiguity. Standard primary-backup replication does not work correctly if network partitions occur. Figure 2 depicts two cases. When a network partition occurs between primary and backup (Fig. 2a), the backup and all clients within the partition of the backup will suspect the primary to have failed leading to both partitions having one primary serving local clients. If only clients but not the backup are disconnected from the primary (Fig. 2b), the system continues to run correctly, but from the disconnected clients' point of view, the system is unavailable.

Quorums One major approach to handle network partitions is to use quorums [7,13]. A read operation has to access a read quorum of replicas while a write operation has to access a write quorum of replicas. A quorum could, e.g., be the majority of replicas. Quorums are defined such that any two write operations on the same data item access at least one common replica. This is needed to serialize write operations and assure that any given write quorum has at least one replica with the latest version of the data item. Each read quorum must overlap with each write quorum. Therefore, each read operation is guaranteed to read at least one replica with the latest version. The beauty of



Replication for High Availability. Figure 2. Network partitions. (a) Two primaries. (b) Unavailability for some clients.

quorums is that they handle both replica failures and network partitions in the same way. It does not matter whether a client cannot reach a particular replica because it is down or disconnected, as long as the client can access a quorum of replicas, the operation can succeed. Recovery does not need any particular actions. While a replica is down it will not participate in any operations. Once it restarts it can be accessed and build part of a quorum. However, its data versions are outdated. Thus, any read operation will not read the data versions of this replica but the most current one from other replicas in the quorum. When the replica participates in a write quorum for a data item, the write operation will automatically update the value of the data item to the latest version.

Despite these attractive properties, quorums have not been used in database systems so far because of the complexity of handling database operations (e.g., complex SELECTs) on a quorum of replicas. The entry *Quorum Systems* discusses quorums in detail.

A possibility is to combine the strengths of ROWAA and quorums [1]. A connected group of replicas processes transactions according to ROWAA if the group fulfills a quorum requirement. A quorum is defined so that it is impossible that two partitioned groups of replicas can fulfill the quorum requirement. Group membership is dynamically adjusted whenever failures, restarts or network reconfigurations occur.

1-Safe (Lazy) Replication in WANs A further approach to address network partitions is to trade data accuracy with availability. It generally follows the ROWAA technique and performs update propagation lazily after commit (i.e., 1-safe). Each read operation only needs to access one copy of the data item, typically the one that is the closest. Thus, queries are usually fast and high availability is given since only one replica needs to be reachable.

For update transactions, one approach is to still have a primary replica and execute all update transactions at the primary. The primary then propagates the updates to all available secondary replicas sometime after commit. Having a single primary for update transactions allows for a globally correct serialization order. Propagating lazily allows the primary to commit a transaction without caring whether the other replicas are available. However, the secondaries can have outdated data, and thus local read operations might read stale data. Furthermore, if a client cannot reach the primary, its update transactions cannot be executed.

A second approach allows an update transaction to execute and commit at any replica, typically the closest, (update everywhere). This provides high availability for update transactions. However, data copies might diverge requiring some form of conflict detection and reconciliation, and read operations can read inconsistent data. Still, for many WAN applications, this is the only feasible option. Also, in mobile environments with planned periods of disconnection, this is the only way to provide availability to the mobile units.

In these lazy strategies, recovery is done incrementally. When a site wants to propagate an update but cannot reach a replica, it stores the update locally on persistent storage. When the replica is again available,

the update will be propagated. That is, propagation is typically done via a persistent queue guaranteeing that an update is propagated and applied exactly once despite transient failures.

Other Failure Types

There exist other failures that are not considered here. For instance message loss can be handled by the underlying communication system.

Key Applications

Basically all commercial database systems provide a high availability solution based on primary-backup replication. Furthermore, other data replication strategies are also often supported and can be deployed for high availability and performance reasons. Most businesses that have update intensive workloads and require 24/7 availability deploy one or more of these solutions.

Outside the database domain, lazy (1-safe) replication is used both for fast local access and availability by file systems [10,12], web-servers [11] and cooperative applications such as distributed calendars.

Replication for fault-tolerance has also become popular in application servers. They represent the middle-tier in modern mutli-tier architectures, and maintain various kinds of data such as session information or cached versions of data stored in persistent storage. The challenges are similar as in pure database replication as the access to this data is transactional [5,14].

Future Directions

As applications are built increasingly on component-based architectures, combining software and hardware from many different vendors, it becomes a challenging task to assure that the system as a whole provides fault-tolerance. According to the end-to-end paradigm, the levels closest to the application might be the only ones able to completely implement the required level of fault-tolerance. On the other hand, lower levels might be able to ensure fault-tolerance more efficiently.

Cross-references

- ▶ Data Replication
- ▶ Logging and Recovery
- ▶ Optimistic Replication and Resolution
- ▶ Quorum Systems
- ▶ Replica Control
- ▶ Traditional Concurrency Control for Replicated Databases
- ▶ Two-Phase Commit
- ▶ WAN Data Replication

Recommended Reading

1. Abbadi A.E. and Toueg S. Availability in partitioned replicated databases. In Proc. 5th ACM SIGACT-SIGMOD Symp. on Principles of Database Systems, 1986, pp. 240–251.
2. Bernstein P.A. and Goodman N. An algorithm for concurrency control and recovery in replicated distributed databases. ACM Trans. Database Syst., 9(4):596–615, 1984.
3. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency control and recovery in database systems. Addison Wesley, Reading, MA, 1987.
4. Budhiraja N., Marzullo K., Schneider F.B., and Toueg S. The primary-backup approach. In Distributed Systems S. Mullender (ed.). (2nd Edition), Addison Wesley, Reading, MA, 1993, pp. 199–216.
5. DeCandia G., Hastorun D., Jampani M., Kakulapati G., Lakshman A., Pilchin A., Sivasubramanian S., Vosshall P., and Vogels W. Dynamo: Amazon's highly available key-value store. In Proc. 21st ACM Symp. on Operating System Principles, 2007, pp. 205–220.
6. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1996, pp. 173–182.
7. Jiménez-Peris R., Patiño-Martínez M., Alonso G., and Kemme B. Are quorums an alternative for data replication? ACM Trans. Database Syst., 28(3):257–294, 2003.
8. Kemme B., Bartoli A., and Babaoglu Ö. Online reconfiguration in replicated databases based on group communication. In Proc. IEEE Int. Conf. on Dependable Systems and Networks, 2001, pp. 117–130.
9. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based data replication providing snapshot isolation. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 419–430.
10. Satyanarayanan M., Kistler J.J., Kumar P., Okasaki M.E., Siegel E.H., and Steere D.C. Coda: A highly available file system for a distributed workstation environment. IEEE Trans. Comput., 39(4):447–459, 1990.
11. Sivasubramanian S., Szymaniak M., Pierre G., and van Steen M. Replication for web hosting systems. ACM Comput. Surv., 36 (3):291–334, 2004.
12. Terry D.B., Theimer M., Petersen K., Demers A.J., Spreitzer M., and Hauser C. Managing update conflicts in Bayou, a weakly connected replicated storage system. In Proc. 15th ACM Symp. on Operating System Principles, 1995, pp. 172–183.
13. Thomas R.H. A majority consensus approach to concurrency control for multiple copy databases. ACM Trans. Database Syst., 4(2):180–209, 1979.
14. Wu H. and Kemme B. Fault-tolerance for stateful application servers in the presence of advanced transactions patterns. In Proc. Int. Symp. on Reliable Distributed Systems, 2005, pp. 95–108.

Replication for Scalability

Ricardo Jiménez-Peris, Marta Patiño-Martínez
Universidad Polytecnica de Madrid, Madrid, Spain

Synonyms

Scale out; Cluster replication; Scalable database replication

Definition

One of the main uses of data replication is to increase the scalability of databases. The idea is to have a cluster (of possibly inexpensive) nodes, to replicate the data across the nodes, and then distribute the load among them. In order to be scalable, the more nodes are added to the system, the higher the achievable throughput should be. The scale reached today is on tens of nodes (i.e., below 100 nodes). Communication is not an issue since CPU and IO overheads are dominant. The approach in the last years has been to learn from the traditional approaches but change some fundamentals so that the limitations of these traditional approaches are avoided.

In order to attain scalability each transaction should not be fully processed by every replica. This depends on how transactions are mapped to replicas. For read only transactions, it is easy to avoid redundant processing since they can be executed at any single replica. Update transactions are more challenging, since updates should be reflected at all replicas where there are copies of the updated data items. This should be achieved in an atomic way across the replicated system what makes scalability challenging. Under the traditional approaches, atomicity was attained by means of distributed locking (for isolation) and two-phase-commit (for failure atomicity), what resulted in solutions lacking scalability.

A second aspect introduced by updates is their overhead. With symmetric update processing no scalability is achieved for updates, since all replicas pay the cost of fully executing the transaction. However, for asymmetric processing some scalability is possible, since only one replica pays the cost of fully executing the transaction, whilst the others just pay the cost of propagating and installing the resulting updated tuples. Another of the major factors that impact the scalability is the degree of replication. Under full replication, all replicas keep a full copy of the database. This means that an update transaction executed at one of the replicas should propagate the resulting updated tuples to all other replicas. This update propagation overhead increases its relative weight with an increasing number of replicas, what inherently limits the scalability. Some research efforts pursue partial replication to overcome the scalability limit of full replication.

There are a number of additional factors that influence scalability. One of such factors is the consistency being provided. The traditional consistency criterion for replicated databases, 1-copy-serializability, constrains the potential concurrency therefore limiting the scalability. A lot of research has targeted to relax consistency in order to increase the scalability.

Historical Background

Traditional textbook approaches for data replication [3] were concerned with consistency and did not provide any scalability. The paper from Gray et al. [9] provided analytical evidence of this lack of scalability triggering a large body of research performed during the last decade on scalable database replication.

One set of approaches that aim at attaining scalability were based on lazy replication [8,4]. In order to provide consistency, some of these lazy approaches constrained update propagation to attain 1-copy-serializability [4].

A second batch of research explored scalable eager (synchronous) replication. The seminal approaches in this direction were Postgres-R [12] and the database state machine [17]. These approaches explored optimistic concurrency control under a white box (or kernel-based) replication approach and triggered the research performed in the last decade around scalable data replication. Due to the complexity of white box approaches, middleware-based replication was proposed to simplify the engineering of replication. The seminal approach to middleware-based replication was Middle-R [22,16] that explored scalable pessimistic replica control. The middleware approach from Middle-R became very popular and today it is one of the main approaches to engineer database replication [1,20,15,5]. Middleware approaches took two different flavors, those based on group communication [22,16,15] and those based on schedulers [1,5,20].

Another batch of research has looked at trading off some consistency in favor of scalability. There are two main families within this batch: those stemming from lazy approaches and those coming from eager replication. On the lazy replication side, consistency

was relaxed through the concept of freshness. Freshness quantifies consistency, e.g., by the potential number of missed updates [8]. On the eager replication side, consistency was relaxed by resorting to isolation levels lower than serializability. The most popular one has been snapshot isolation [13] provides full 1-copy correctness (1-copy-snapshot-isolation, 1CSI) for snapshot isolation replicated databases. 1CSI is provided by a number of replication protocols both under update-everywhere [13] and primary-copy [20]. Another consistency criterion proposed based on snapshot isolation is Generalized Snapshot Isolation (GSI) [7]. It enables queries to see older snapshots that are prefix consistent. More recently, snapshot isolation has also been exploited in the context of lazy approaches in [6] and also for multi-tier architectures [18].

The fourth batch of research has tried to overcome the lack of scalability of full replication. The paper [11] demonstrated analytically the scalability limits of full replication. Two ways to overcome this inherent scalability limitation are quorum-based replication and partial replication. Quorum-based replication can improve the scalability for extreme update workloads, but introduces a number of technical problems, the most important ones related to the scalability of predicate reads [11]. Partial replication approaches have aimed at improving scalability by limiting the number of copies of each data item [21,19].

Foundations

One of the uses of database replication is to increase the scalability of the system. The use of data replication to scale is also known as *scale-out* approach, in which the scalability increases by adding new nodes. This contrasts with the *scale-up* approach in which a system scales by substituting the current node with a more powerful computer. This section discusses the major factors that influence scalability of data replication, namely: how to attain atomicity, the mapping of transactions to replicas, and the consistency criterion.

Attaining Atomicity and Isolation

The traditional way to attain transactional properties in data replication [3] was based on using distributed locking to serialize transactions in the same order (isolation at replicated level) and using two-phase-commit (2PC) to guarantee that either all replicas commit the transaction or none (failure atomicity). Gray et al. [9] showed analytically the lack of scalability of traditional approaches. Basically, distributed locking has a probability of deadlock that increases exponentially with the number of transactions and 2PC does not scale and produces high response times. Two trends were followed to escape from the bottlenecks of traditional approaches. Lazy protocols commit transactions without waiting for transaction update propagation, that is, updates are propagated lazily or asynchronously with respect to transaction commit at the replica that processed the transaction. Alternatively, eager protocols propagate transaction updates atomically using alternative ways to enforce atomicity more scalable than 2PC.

Lazy protocols have mainly resorted to two approaches to attain some levels of consistency: primary-copy (aka single-master) and multi-master data replication. The *primary-copy* approach simplifies the consistency problem by enabling updates only at a single replica known as primary. All the other replicas, known as secondaries, only allow the execution of read-only queries. In this way, the consistency problem boils down to installing the updates from the primary in FIFO order or a more relaxed order that guarantees the same relative order of conflictive transactions. A popular replication protocol providing primary-copy is [20].

Primary-copy replication has its bottleneck in the primary that has to fully process all update transactions. As an alternative, *multi-master* replication has been proposed in which update transactions can be processed by any replica. Different approaches have been adopted for multi-master replication. In some approaches the way update transactions are propagated is constrained to guarantee high levels of consistency [4]. In others consistency is relaxed by providing quantitative levels of consistency such as freshness. A more detailed discussion on freshness is provided later in this text. Leganet is one of the systems with a lazy replication protocol providing freshness [8].

Eager protocols aim at high levels of qualitative consistency. Most eager protocols are either based on group communication or on a scheduler. *Approaches based on group communication* use atomic multicast to enforce atomicity in a more scalable way than 2PC. Atomic multicast provides failure atomicity by guaranteeing that all or none of the replicas receive messages and also guarantees that all replicas receive the messages in the same relative order. In this way, each replica is able to schedule the messages containing updates in an order consistent with respect to the common total order. There are a number of data replication protocols based on

group communication such as Postgres-R [12], database state machine [17], and Middle-R [16,13].

The *scheduler-based approach* has also been proposed for attaining consistency in eager protocols. The idea is to use a node as transaction scheduler. This scheduler is in charge of enforcing consistency. It labels transactions with a sequence number and forwards them to the replicas. In this way, replicas are able to order conflictive transactions in the same way and atomicity can also be enforced by preventing gaps in the transaction processing. Some of the database replication systems based on scheduler are conflict-aware data replication [1] and C-JDBC [5].

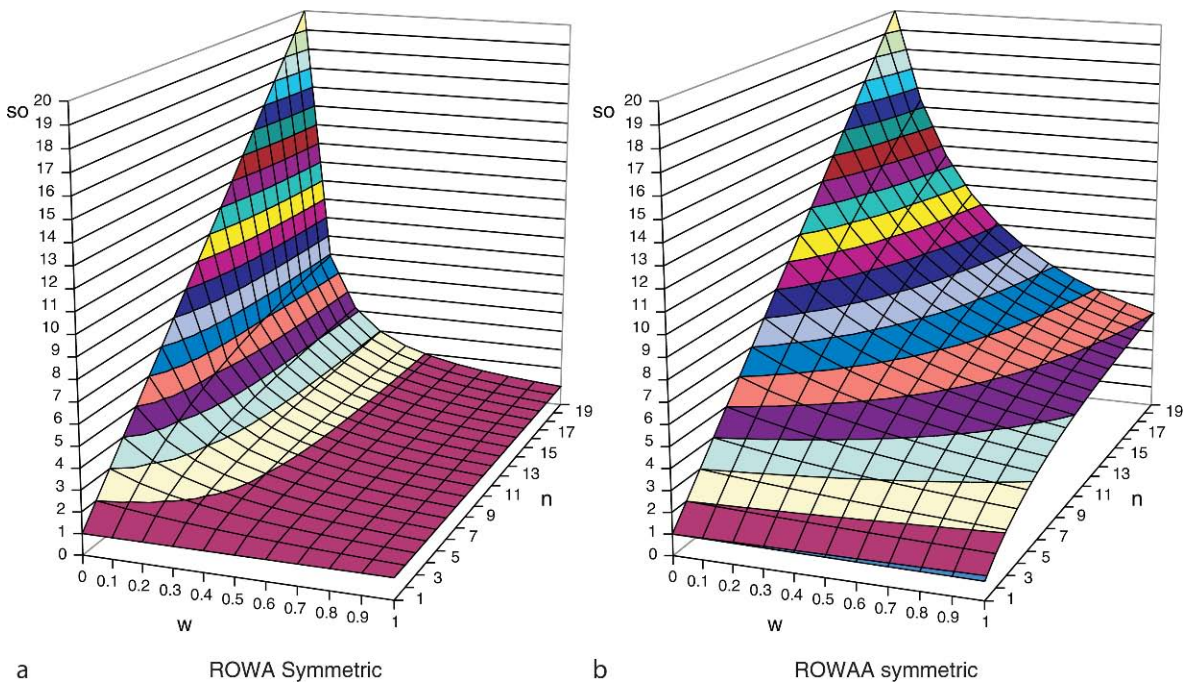
Transaction Mapping

The mapping of transactions to replicas is crucial in attaining scalability. There are several issues related to the transaction mapping: which replicas processes read-only transactions (queries), how update transactions are processed, and how many copies of each data item are kept.

The first issue is *how read-only transactions (queries) are managed*. Since queries only read data, for fully replicated systems it becomes possible to execute queries at any single replica, whilst update transactions are fully executed at all replicas. This mapping is known as read-one write-all-available approach (ROWAA) [3]. It enables to scale under read workloads, since for reads the load is shared among replicas. However, this approach fails to scale even with a small percentage of update transactions [11]. ROWAA has been used by the early replication protocols that are surveyed in [3].

The second issue is *how update transactions are processed*. Update transactions can be executed at only one of the replicas, as far as the other replicas get the resulting updates and install them. This mapping is known as *asymmetric update processing*, in contrast with *symmetric update processing* in which update transactions are fully executed by all replicas. This mapping also enables sharing the load introduced by update transactions, enabling scalability with update workloads [11].

Figure 1 compares the scalability of symmetric and asymmetric processing under ROWAA. The x-axis shows the fraction of writes (1.0 = 100% writes), the y-axis shows the number of nodes. The z-axis shows the scale-out, i.e., how many times the throughput of a single-node system is multiplied by the replicated system. Figure 1 shows that symmetric processing only achieves a high scale-out for very low values of w. With asymmetric processing, assuming that installing updates has around 15% of the costs of executing



Replication for Scalability. Figure 1. Scalability of symmetric and asymmetric processing (wo = 0.15 for asymmetric).

the update transaction itself ($wo = 0.15$), the scale-out is relatively high even for higher update rates.

A different mapping that has been proposed to further increase the scalability of update workloads is based on quorum systems. Quorum systems enable to write just in a subset of the replicas (write quorum). This has the associated tradeoff that reads should also be performed on a subset of replicas (read quorum). Quorums can compete with ROWAA when the percentage of updates in the workload is very high (80–100%) [11]. However, they only work for transactions accessing individual objects. When combined with collections of objects such as tables, queries become too expensive for being practical.

The third issue is *how many copies of each data item are kept (aka degree of replication)*. The paper [11] demonstrated analytically the scalability limits of full replication as a function of the ratio of the cost of fully executing the transaction and installing the resulting updates termed write overhead, *wo*.

With typical write overheads of 0.15 and below, the scalability becomes very reasonable for a few tens of nodes. However, full replication does not scale beyond that. The reason is that the overhead introduced by the update propagation and installation consumes higher and higher fractions of the capacity of the full system what finally is translated in consuming any extra additional capacity to process updates from the existing replicas.

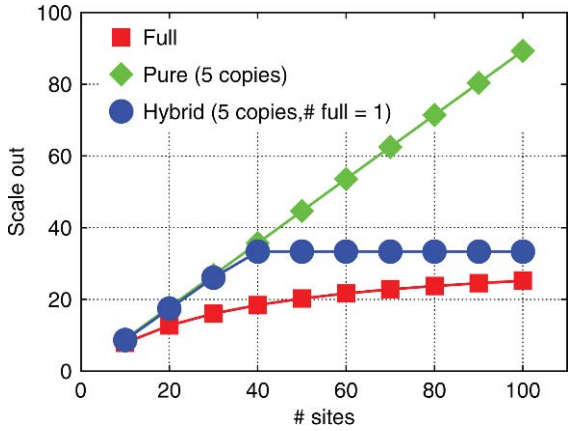
The alternative to scale beyond the limits of full replication is to use partial replication. Partial replication, however, also introduces other challenges. If there is no single replica with the full database, what is called pure partial replication, transactions become inherently distributed. As an alternative, if there is application knowledge available, it can be exploited to guarantee that there will be at least a replica that can execute a transaction fully locally. A potential solution is to have hybrid partial replication in which there are some replicas containing the full database and some replicas storing a fraction of the database. The former enables to avoid distributed transactions, and the latter enables to scale. Unfortunately, this approach, although it scales significantly better than full replication, reaches a limit of scalability. This limit is reached when the full replicas are saturated with update installation from partial replicas (even if they do not process any local transaction). The way to scale beyond hybrid partial replication is to use pure partial replication, what implies to be able to process replicated distributed transactions in an efficient way. The scalability of partial replication, both hybrid and pure, has been studied recently both analytically and empirically [21].

Figure 2 compares the scale-out of full replication, pure partial replication where each data item has five copies equally distributed among the nodes, and hybrid replication where one node has a copy of the entire database, and each data item has a total of five copies. The update workload in this case is 20% one can clearly see, that only pure replication has no scalability limit while the others are limited by the fact that all updates have to be performed on all copies, and there is at least one server that has copies of all data items.

Consistency Criterion

Another of the major factors influencing the scalability of replication is the consistency criterion. Some replication protocols provide a qualitative consistency. These protocols are usually *eager* protocols in which replica control is tightly integrated with update propagation to guarantee the qualitative consistency criterion. Other protocols have aimed at *quantitative consistency*, typically *lazy* protocols. In lazy protocols replica control is more relaxed and typically offers quantitative consistency, and in some cases simply eventual consistency.

Qualitative consistency for data replication provides a formal definition of the attained consistency. The consistency criterion can be seen as the extension of the



Replication for Scalability. Figure 2. Scalability of full versus partial data replication.

isolation concept of centralized systems to a replicated setting. Isolation formalizes the consistency for concurrent executions in a centralized system. The replication consistency criterion extends this formal notion to the consistency of the concurrent transaction execution in a replicated system. The traditional criterion for replicated databases has been 1-copy-serializability (1CS) [3]. This criterion states that the concurrent execution of a set of transactions in a replicated system should be equivalent to the execution in a serializable centralized system. 1CS has been the only criterion used till very recently. 1CS, as its centralized counterpart, serializability, constrains the potential concurrency in the system by making reads and writes conflicting. Constraining the potential concurrency is harmful for scalability (even for performance in a centralized system) since it might prevent to fully utilize the available capacity in the replicated system. In order to overcome the scalability limitation of 1CS, some researchers have explored snapshot isolation (SI) as isolation notion on which to build a replication consistency criterion. 1-copy snapshot-isolation [13] (1CSI) provides the notion of 1-copy correctness underlying in 1CS for a SI database. That is, the concurrent execution of a set of transactions in a replicated system should be equivalent to a centralized (1-copy) execution in a centralized SI database. The main advantage of SI is that reads and writes do not conflict. The only conflicts are between writes on the same tuples that are rare in most applications. 1CSI enables replica control protocols that are based on SI databases. This means that they do not have contention problems due to read-write conflicts, what results in higher scalability. Snapshot isolation is currently being used for different replica control protocols, lazy primary-copy replication [20], eager update-everywhere [13], lazy replication [6], and application server replication [18].

Quantitative consistency aims at increasing the scalability by relaxing the consistency enabling queries to see outdated data bounded quantitatively. It is typically used in lazy propagation schemes. Freshness criteria bound how much the value a query reads differs from the actual value of the data item. Freshness, e.g., could bound the number of updates a query has missed or the time since the copy read was last updated. The concept of freshness has been exploited by many different systems, and has been used in the context of primary-copy [20], multi-master [8], or application server replication [2]. The entry *Consistency Models for Replicated Data* provides a more detailed discussion on the different qualitative and quantitative consistency levels that exist.

Key Applications

Database replication has a wide number of applications. In principle any database that reaches saturation can be substituted by a replicated database to scale out. *Enterprise data centers* are certainly one of the main targets of database replication in order to scale for high loads typical of these systems. *Web farms* hosting dynamic content require the use of a database for storing it. In this kind of systems the main bottleneck is precisely the database and therefore, it can benefit from replicated databases. Grid systems are able to scale for applications based on the paradigm of bag of tasks, problems that can be split in a myriad of small subproblems that can be solved independently. In some cases, some grid applications require database access that is what finally becomes the bottleneck of the system. By employing a replicated database as a *data grid*, these grid applications can increase their scalability. A recent and increasingly important trend is *Software as a Service (SaaS)*. In this new paradigm, applications are hosted at a remote data center such as Google. SaaS aims for providing transparent scalability by means of self-provisioning. In this way, the hosted application can increase the number of replicas as needed depending on the received load and paying only for the resources it needs. SaaS platforms require an underlying scalable storage system such as a replicated database. Another area in which database replication is becoming a competitive solution is edge computing. *Edge computing* aims at moving web contents closer to clients located at the edge of Internet. Centralized solutions to edge computing are not adequate since distant clients observe high latencies. With edge computing, the contents is cached or replicated at data centers geographically close to the client to mask the latency. Early approaches only managed to reduce the latency of static contents. Recently, it has been shown that by using database replication in wide area networks it becomes possible also to mask the latency for dynamic contents [23].

Cross-references

▶ Data Replication

Recommended Reading

1. Amza C., Cox A.L., and Zwaenepoel W. Distributed versioning: consistent replication for scaling back-end databases of dynamic

content web sites. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2003.
2. Bernstein P.A., Fekete A., Guo H., Ramakrishnan R., and Tamma P. Relaxed-currency serializability for middle-tier caching and replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 599–610.
3. Bernstein P.A., Hadzilacos V., and Goodman N. Concurrency Control and Recovery in Database Systems. Addison Wesley, 1987.
4. Breitbart Y., Komondoor R., Rastogi R., Seshadri S., and Silberschatz A. Update propagation protocols for replicated databases. In Poc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
5. Cecchet E., Marguerite J., and Zwaenepoel W. C-JDBC: flexible database clustering middleware. In Proc. USENIX 2004 Annual Technical Conference, 2004.
6. Daudjee K. and Salem K. Lazy database replication with snapshot isolation. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 715–726.
7. Elnikety S., Zwaenepoel W., and Pedone F. Database replication using generalized snapshot isolation. In Proc. 24th Symp. on Reliable Distributed Syst., 2005, pp. 73–84.
8. Gançarski S., Naacke H., Pacitti E., and Valduriez P. The leganet system: freshness-aware transaction routing in a database cluster. Inf. Syst., 32(2):320–343, 2007.
9. Gray J., Helland P., O'Neil P., and Shasha D. The dangers of replication and a solution. In Poc. ACM SIGMOD Int. Conf. on Management of Data, 1996.
10. Jiménez-Peris R., Patiño-Martínez M., Alonso G., Kemme B. Scalable Database replication middleware. In Proc. 22nd Int. Conf. on Distributed Computing Systems, 2002.
11. Jiménez-Peris R., Patiño-Martínez M., Alonso G., and Kemme B. Are quorums an alternative for data replication. ACM Trans. Database Syst., 28(3):257–294, 2003.
12. Kemme B. and Alonso G. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
13. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based data replication providing snapshot isolation. In Poc. ACM SIGMOD Int. Conf. on Management of Data, 2005.
14. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Enhancing edge computing with database replication. In Proc. 26th Symp. on Reliable Distributed Syst., 2007.
15. Muñoz-Escoí F.D., Pla-Civera J., Ruiz-Fuertes M.I., Irún-Briz L., Decker H., Armendáriz-Iñigo J.E., and de Mendívil J.R.G. Managing transaction conflicts in middleware-based database replication architectures. In Proc. 25th Symp. on Reliable Distributed Syst., 2006, pp. 401–420.
16. Patiño-Martínez M., Jiménez-Peris R., Kemme B., and Alonso G. Middle-R: consistent database replication at the middleware level. ACM Trans. Computer Syst., 23(4):375–423, 2005.
17. Pedone F., Guerraoui R., and Schiper A. The database state machine approach. Distributed and Parallel Databases, 14(1): 71–98, 2003.
18. Perez-Sorrosal F., Patiño-Martínez M., Jiménez-Peris R., and Kemme B. Consistent and scalable cache replication for multi--tier J2EE applications. In Proc. ACM/IFIP/USENIX 8th Int. Middleware Conf., 2007, pp. 328–347.
19. Pinto A.L., Oliveira R., Moura F., and Pedone F. Partial replication in the database state machine. In IEEE International Symposium on Networking Computing and Applications, 2001, pp. 298–309.
20. Plattner C. and Alonso G. Ganymed: scalable replication for transactional web applications. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2004.
21. Serrano D., Patiño-Martínez M., Jiménez-Peris R., and Kemme B. Boosting database replication scalability through partial replication and 1-copy-snapshot-isolation. In IEEE Pacific Rim Dependable Computing Conference, 2007, pp. 328–347.
22. Serrano D., Patiño-Martínez M., Jiménez-Peris R., Kemme B. An Autonomic Approach for Replication of Internet-based services. In Proc. 27th Symp. on Reliable Distributed Syst., 2008.

Replication in Multi-Tier Architectures

Ricardo Jimenez-Peris, Marta Patiño-Martinez
Universidad Polytecnica de Madrid, Madrid, Spain

Synonyms

Cloud computing; Scale out; Cluster replication; Scalable replication; Application server clustering; SOA replication

Definition

Modern middleware systems are commonly used in multi-tier architectures to enable separation of concerns. For each tier, a specific component container is provided, tailored to its mission, web interface, business logic, or persistent storage. Data consistency across tiers is guaranteed by means of transactions. This entry focuses on the main three tiers: web, application server, and database tiers.

Middleware systems are at the core of enterprise information systems. For this reason they require high levels of availability and scalability. Replication is the main technique to achieve these two properties. First middleware replication approaches addressed the replication of individual tiers and focused initially on providing availability and later, on scalability. However, an integral approach is needed to provide availability and scalability of multi-tier systems. Replicating an individual tier increases its availability and might increase its scalability as well. However, replicating a single tier is not sufficient for providing data availability and scaling for the whole system; the non-replicated tiers

eventually become a single point of failure and/or a potential bottleneck. A potential solution is to combine independently replicated tiers. However, due to their independent replication, they are not aware of the replication of each other; this results in inconsistency problems in the event of failures.

Recent research has introduced new approaches to replication of multi-tier architectures. The two main replication architectures are horizontal and vertical replication. In horizontal replication, the tiers are replicated independently, then measures are taken to make at least one of them aware of the other to deal in a consistent way with failures and failovers. In vertical replication, a set of tiers (typically the application and database tiers) are replicated as a unit, and the replication logic is encapsulated in the upper tier (typically the application server). Another important aspect of replication of multi-tier systems is related to the definition of the correctness criterion, i.e., the conditions a replicated multi-tier architecture should fulfill in order to provide consistency. There are two important criteria in this area, namely, exactly-once semantics and 1-copy correctness.

Historical Background

The first wave of research on the replication of multi-tier architectures focused solely on the availability of individual tiers, mainly the application server and database tiers. In the first wave, the theoretical basis for process replication was set [22]. The state machine approach was proposed to guarantee the consistency of replicated servers. This approach stated that a server could be consistently replicated by guaranteeing that all server replicas receive the requests in the same order and the server deterministically processed them. This determinism requirement was interpreted as the server had to be sequential, which was too restrictive for real servers that are multi-threaded in nature.

The efforts on the database tier concentrated on how to attain data consistency in the advent of failures and how to characterize data consistency. These initial efforts led to lock-based data replication approaches. These approaches were not scalable but they enabled the development of correctness criterion. The criterion characterizing correctness was 1-copy-serializability, i.e., replication should be semantically transparent, the replicated database should allow only those executions that had an equivalent in a non-replicated (1-copy) system.

A second wave of research on middleware replication addressed the fault-tolerance of real systems in the context of CORBA, which led to the (fault-tolerant) FT-CORBA standard. These approaches were based on the theoretical foundations of the state machine and tried to solve the difficulties of engineering replication in real middleware systems [2,7,16]. A new line of research opened in this wave enabled the use of the state machine approach for multi-threaded servers. A seminal paper [11] showed that it was possible to devise a scheduler that can guarantee deterministic behavior of multi-threaded servers. Later approaches studied how to increase the potential concurrency [3].

The efforts on the database tier focused on how to attain scalability. Some approaches relaxed consistency in the quest for scalability relying on lazy replication approaches (see the entry *Replication for Scalability* for a deeper view) [5,17]. Other approaches aimed at attaining scalability while preserving full consistency [1,12,18,19]. The three later approaches leveraged group communication to provide consistency in any failure scenario.

The third wave of research on middleware replication focused on dealing with specific aspects of multi-tier architectures, more concretely, on how to deal with transactional processing in a consistent way, the consistency criteria, and how to attain scalability. The issue of the lack of transactional consistency in replication approaches for multi-tier architectures was then raised [8]. In the context of FT-CORBA, it was studied how to connect a replicated CORBA application server with a shared database guaranteeing exactly once semantics [25], therefore providing the same semantics as the non-replicated system. Also in the J2EE context transactional consistency has been studied. First approaches replicated session state in the application server replicas sharing a common database [24] and enforced transactional consistency in this context.

More recently, replication of multiple tiers has also been studied, e.g., in a vertical replication approach [21]. That is, an application server and database server pair is the replication unit. Database servers are centralized and are not aware of the replication. The application server encapsulates the replication logic transparently to the database. The application server replicas interact among them to enforce consistency (i.e., 1-copy-serializability). Additionally, in this approach transactions are not aborted in the advent of failures from the client perspective, that is, it provides high transaction availability. Another recent approach has considered the issue of replicating application

servers accessing multiple databases [14]. In this approach a primary-backup model is adopted. Clients interact with the primary application server. The most general case in which the application server accesses multiple databases is considered. This case is more complex since it involves dealing with two-phase-commit (2PC) and the failover of the 2PC coordinator (the transaction manager of the application server). The primary application server accesses multiple databases to process client requests. When the client transaction ends, 2PC is started and coordinated by the primary application server. The primary checkpoints session state as well as transaction management information to the backups. The approach is able to handle the primary failure and enforce transaction consistency for distributed transactions, providing transaction manager (2PC coordinator) failover.

On the correctness side, it was studied how to guarantee exactly once semantics in multi-tier architectures in which the application server tier is stateless [9]. That is, how to guarantee transaction atomicity and exactly once semantics in the advent of a failover in replicas of stateless application servers. A thorough study of the different approaches for multi-tier replication that guarantee transactional consistency was presented in [13].

On the database tier, the third wave of research has focused on boosting the scalability beyond a few tens of replicas attained in the second wave. Two aspects were mainly addressed: How to overcome the scalability limits of 1-copy-serializability and full replication. 1-copy-serializability limits significantly the potential concurrency of the system which had an impact on the attainable scalability. This resulted in approaches based on a more relaxed consistency criterion, 1-copy-snapshot-isolation [15]. The scalability limits of full replication were studied analytically in [10] which led to the exploration of partial replication. Partial replication has resulted in boosting the scalability of full data replication [23], although it has some complexities such as how to deal with data partitioning and distributed transactions.

Nowadays, integral approaches for the replication of multi-tier architectures are becoming common. Research in the area is aiming at enhancing the scalability of the seminal approaches to replication of multi-tier systems. More concretely [4] studies how to replicate the application server tier with a shared database and boost the scalability by relaxing data currency through freshness constraints. Another recent approach is looking at the scalable replication of application server and database tiers [20]. In order to overcome the scalability bottleneck of serializability, this approach is based on 1-copy-snapshot isolation. Additionally, it deals for the first time with the issue of the cache consistency at the application server for relaxed isolation levels, in particular, for snapshot isolation.

Foundations

This section reviews the main replication approaches for the different tiers. Both approaches (replicating a single tier and replicating multiple tiers) will be considered.

The web tier is simple to replicate due to its stateless nature. On the other hand, it has to deal with a very specific aspect that lies in the availability and scalability of the connectivity to the Internet. This has resulted in multiple approaches to virtualize URLs and IP addresses. IP virtualization enables providing transparent failover since the client sees a single logical IP despite the fact that two physical IPs might get involved in the processing of its requests upon a failover. IP virtualization also enables to provide transparent load balancing since a network switch can transparently route requests to a logical IP to different physical IPs to balance the load across different sites. A thorough survey on the replication of the web tier can be found in [6].

The replication of the application server tier is more challenging due to its stateful nature and also due to its support for transactional processing. It is possible to distinguish between replication for availability and replication for scalability. Replication for availability aims at providing fault-tolerance but typically without taking care of scalability, even providing negative scalability (a performance below the one of the non-replicated server). Replication for scalability, in addition to providing availability, it is able to increase the system capacity by increasing the number of replicas.

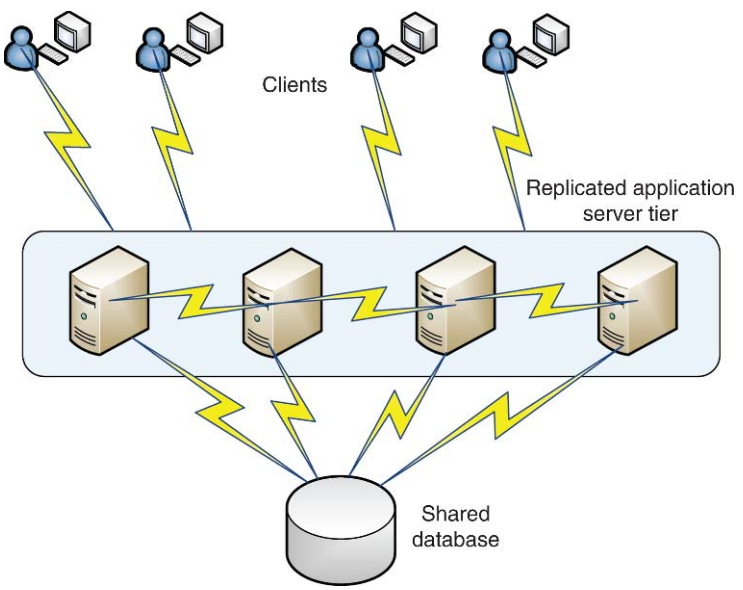
Replication for availability approaches rely on the state machine approach [22]. The state machine replication is based on two basic principles: (i) All replicas receive requests in the same global order; (ii) Each replica behaves deterministically and then, given the same input request sequence produces the same output sequence. Therefore, replication protocols for availability rely on a method for ordering client requests in the same

order at all replicas to attain the required global order, typically, using group communication and total order multicast. Then, they remove the non-determinism from the application. In this class, it is possible to find all the efforts around FT-CORBA such as Eternal [16].

One of the main sources of non-determinism is multi-threading. Most of the approaches simply tackle with single-threaded (sequential) servers to guarantee determinism. However, multi-threading is a necessary feature for real-world servers. Some research has been conducted in how to guarantee determinism of replicas in the presence of multi-threading [3,11]. This research has produced different multi-threading schedulers that enforce deterministic scheduling enabling the replication of multi-threaded servers. The seminal approach [11] showed that by using a deterministic scheduler and setting a deterministic method to schedule requests integrated in the deterministic scheduler it became possible to guarantee the determinism of multi-threaded servers. Later on some other deterministic schedulers have been proposed aiming at increasing the real concurrency [3].

An important issue that needs to be dealt with in the replication of application servers is how to preserve transactional consistency [8]. The paper [9] proposes e-transactions to preserve consistency and identifies exactly once semantics as a key consistency criterion for replication of transactional multi-tier architectures. e-Transactions also provide a protocol for satisfying exactly-once semantics for replicated stateless application server tiers with a shared database (see Fig. 1). Exactly-once semantics states that each transaction should be executed exactly once despite failures and re-executions due to failovers.

A more evolved approach [24] deals with the replication of stateful application servers also with a shared database. In this approach, the relationship between request and transaction is not constrained to be 1 to 1 as in e-transactions, instead, it can be arbitrary. That is, it can be 1:1, N:1, and 1:N. In [24] it is proposed a protocol providing exactly-once semantics [9] despite failures. The protocol intercepts database accesses labeling them with a global sequence identifier and the associated client and request identifier. Each database requests is multicast to the other replicas with the associated information. In the advent of a failure of the primary replica of the application server, a backup replica will take over. Those clients that have not received a reply to their last request will resubmit it to the new primary. The new primary will execute resubmitted requests intercepting the requests submitted to the database. This interception takes care of executing the database requests in the same order as in the old primary in order to guarantee a



Replication in Multi-Tier Architectures. Figure 1. Horizontal replication with a replicated application server tier and a shared database.

deterministic execution and exactly once semantics. New requests will be delayed till the requests received from the former primary are replayed.

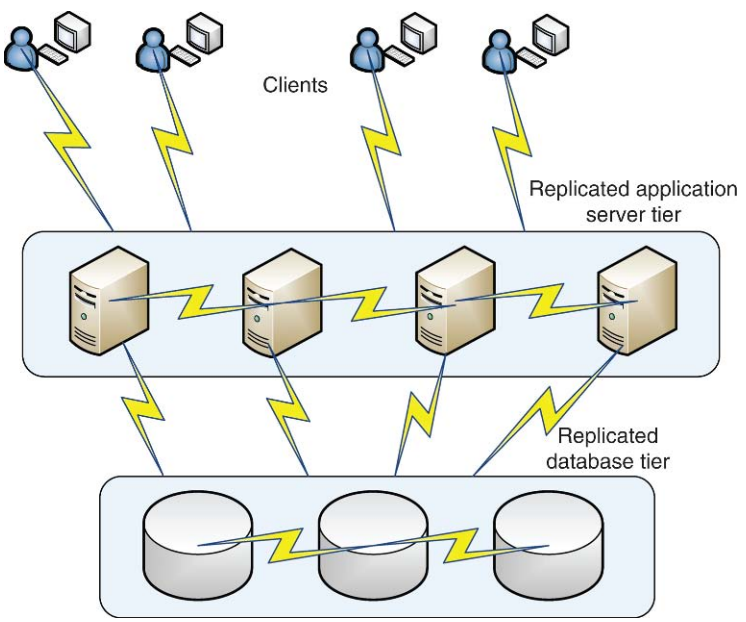
All the aforementioned approaches deal exclusively with the replication of the application server tier, which results in the database becoming a single point of failure and performance bottleneck. More recently, research has been conducted to deal with the replication of both the application server and database tiers guaranteeing consistency despite failures. Two alternative architectures have been considered: horizontal and vertical replication [13].

In horizontal replication each tier is replicated independently (see Fig. 2). However, the integration of two independently replicated tiers results in inconsistency problems in the advent of failures [13]. In order to avoid inconsistencies the replicated tiers sholud become aware of each other to handle failovers consistently. However, this is typically unrealistic since it implies the cooperation of two different vendors very often competitors in the market. [13] proposes two solutions to attain consistent failover by incorporating awareness of replication in only one of the two tiers, either the application server or the database tier.

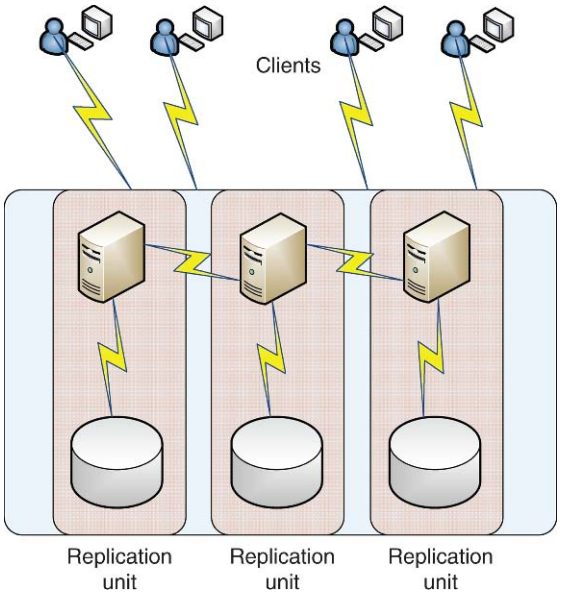
Vertical replication lies in taking as replication unit an application server - database server pair [21] (see Fig. 3). The database server is not aware of the replication. The application server encapsulates all the replication logic. In the vertical replication approach [21] both changes to session (session beans in the context of J2EE) and persistent state (entity beans in the context of J2EE) are captured. The two kinds of changes are propagated to the backup replicas at the end of each client request to provide *highly available transactions*, that is, transactions that do not abort despite failovers.

Both horizontal and vertical replication architectures are also used to attain scalability. Horizontal replication has mainly focused on replicating the middleware tier to replicate sessions while sharing a single database. In these approaches, consistency is attained through the shared database that serializes concurrent database accesses from different replicas. The replication of sessions enables to increase the scalability of applications in which the bottleneck is located in the application server, they are CPU intensive (e.g., image processing). However, these approaches fail to scale when the bottleneck is on the database, which is the most common case in practice.

Vertical approaches have aimed at increasing the scalability independently of whether they are application server or database intensive whilst providing strong consistency [4,20]. To date there are two different approaches to scalable vertical replication: [20] that provides strong consistency and [4] that provides relaxed consistency.



Replication in Multi-Tier Architectures. Figure 2. Horizontal replication with two replicated tiers.



Replication in Multi-Tier Architectures. Figure 3. Vertical replication. Replicated application server and database tiers.

[4] uses a master-slave approach and aims at scaling read-intensive workloads. The master processes all update transactions that get reflected in the underlying database and takes care of propagating a sequence of committed updates to the slaves. Each slave updates the cache and also commits the updates to the database in commit order. Read-only queries can be executed at any replica and can read stale data. The system can be configured to set the staleness or freshness upon which each transaction should be executed.

On the other hand, [20] is an update-everywhere approach (all replicas can process update transactions) and aims at scaling both read and write workloads. Unlike [4,20], provides strong consistency, namely, 1-copy-snapshot-isolation. [20] firstly provides a consistency criterion for application server caching, named cache-transparency. An application server is cache-transparent if it allows the same executions as the application server with the cache disabled. Cache transparency extends the isolation notion of databases to a multi-tier architecture. This is crucial since current application servers may work incorrectly with databases running on some relaxed isolation levels, such as, snapshot isolation. Secondly, it guarantees 1-copy correctness for the multi-tier server which provides replication transparency. [20] builds on 1-copy-snapshot isolation to obtain higher scalability levels since 1-copy-serializability has some strict scalability limits as discussed earlier. Basically, [20] uses multi-versioning in the cache of the application server replicas that is synchronized with the snapshots of the underlying database and among replicas in order to provide cache and replication transparency, respectively.

Cross-references

► Data Replication

Recommended Reading

1. Amza C., Cox A.L., and Zwaenepoel W. Distributed versioning: consistent replication for scaling back-end databases of dynamic content web sites. In Proc. ACM/IFIP/USENIX Int. Middleware Conf., 2003.
2. Baldoni R. and Marchetti C. Three-tier replication for FT-CORBA infrastructures. Software Practice & Experience, 33(8):767–797, 2003.
3. Basile C., Kalbarczyk Z., and Iyer R.K. Active Replication of Multithreaded Applications. IEEE Trans. Parallel and Dist. Syst., 17(5):448–465, 2006.
4. Bernstein P.A., Fekete A., Guo H., Ramakrishnan R., and Tamma P. Relaxed-currency serializability for middle-tier caching and replication. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2006, pp. 599–610.
5. Breitbart Y., Komondoor R., Rastogi R., Seshadri S., and Silberschatz A. Update propagation protocols for replicated databases. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 1999.
6. Cardellini V., Casalicchio E., Colajanni M., and Yu P.S. The state of the art in locally distributed Web-server systems. ACM Comput. Surv., 34(2):263–311, 2002.
7. Felber P., Guerraoui R., and Schiper A. The Implementation of a CORBA Object Group Service. Theory & Practice of Object Systems, 4(2):93–105, 1998.
8. Felber P. and Narasimhan P. Reconciling Replication and Transactions for the End-to-End Reliability of CORBA Applications. In Proc. Int. Symp. on Distributed Objects and Applications, 2002.
9. Frølund S. and Guerraoui R. e-Transactions: End-to-End Reliability for Three-Tier Architectures. IEEE Trans. Software Eng., 28(4):378–395, 2002.
10. Jiménez-Peris R., Patiño-Martínez M., Alonso G., and Kemme B. Are Quorums an Alternative for Data Replication. ACM Trans. Database Syst., 28 (3), 2003.
11. Jiménez-Peris R., Patiño-Martínez M., and Arevalo S. Deterministic Scheduling for Transactional Multithreaded Replicas. In Proc. IEEE Int. Symp. on Reliable Distributed Systems, 2000, pp. 164–173.
12. Kemme B. and Alonso G. Don't be lazy, be consistent: Postgres-R, a new way to implement database replication. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
13. Kemme B., Jiménez-Peris R.R., Patiño-Martínez M.M., and Salas J. Exactly once interaction in a multi-tier architecture. In Proc. of

VLDB Workshop on Design, Implementation and Deployment of Database Replication, 2005.
14. Kistijantoro A.I., Morgan G., and Shrivastava S.K. Enhancing an Application Server to Support Available Components. IEEE Trans. Software Eng., SE-34(4):531–545, 2008.
15. Lin Y., Kemme B., Patiño-Martínez M., and Jiménez-Peris R. Middleware based Data Replication providing Snapshot Isolation. In Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD), 2005.
16. Moser L.E., Melliar-Smith P.M., Narasimhan P., Tewksbury L., and Kalogeraki V. The eternal system: an architecture for enterprise applications. In Proc. Int. Enterprise Distributed Object Computing Conf., 1999, pp. 214–222.
17. Pacitti E. and Simon E. Update propagation strategies to improve freshness in lazy master replicated databases. VLDB J., 8(3), 2000.
18. Patiño-Martínez M., Jiménez-Peris R., Kemme B., and Alonso G. Middle-R: Consistent Database Replication at the Middleware Level. ACM Trans. Computer Syst., 23(4):375–423, 2005.
19. Pedone F., Guerraoui R., and Schiper A. The Database State Machine Approach. Distrib. Parall. Databases, 14(1):71–98, 2003.
20. Perez-Sorrosal F., Patiño-Martínez M., Jiménez-Peris R., and Kemme B. Consistent and scalable cache replication for multi-tier J2EE applications. In Proc. ACM/IFIP/USENIX 8th Int. Middleware Conf., 2007, pp. 328–347.
21. Perez-Sorrosal F., Patiño-Martínez M., Jiménez-Peris R., and Vuckovic J. Highly available long running transactions and activities for J2EE applications. In Proc. 23rd Int. Conf. on Distributed Computing Systems, 2006.
22. Schneider F.B. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. ACM Comput. Surv., 22 (4):299–319, 1990.
23. 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, pp. 328–347.
24. Wu H. and Kemme B. Fault-tolerance for Stateful Application Servers in the Presence of Advanced Transactions Patterns. In Proc. IEEE Int. Symp. on Reliable Distributed Systems, 2005, pp. 95–108.
25. Zhao W., Moser L.E., and Melliar-Smith P.M. Unification of Transactions and Replication in Three-Tier Architectures Based on CORBA. IEEE Transactions on Dependable and Secure Computing, 2(1), 2005.

Report Writing

► Summarization

Representation

► Icon

Reputation

► Trust in Blogosphere

Reputation and Trust

Zoran Despotovic
NTT DoCoMo Communications Laboratories Europe, Munich, Germany

Synonyms

Feedback systems; Word of mouth

Definition

Trust means reliance on something or someone's action. As such, it necessarily involves risks on the side of the subject of trust, i.e., trustor. The main goal of a trust management system is to reduce the involved risks. Reputation systems present a possible solution to do that. They use relevant information about the participants' past behavior (feedback) to encourage trustworthy behavior in the community in question. The key presumptions of a reputation system are that the participants of the considered online community engage in repeated interactions and that the information about their past doings is informative of their future performance and as such will influence it. Thus, collecting, processing, and disseminating the feedback about the participants' past behavior is expected to boost their trustworthiness.

Key Points

The goal of a reputation system is to encourage trustworthy behavior. It is up to the system designer to define what trustworthy means in her specific setting. There are two possible ways to do this: *signaling and sanctioning reputation systems*. In a signaling reputation system, the interacting entities are presented with signals of what can go wrong in the interactions if they behave in specific ways. Having appropriate signals, the entities should decide what behavior is most appropriate for them. An important assumption of the signaling reputation systems is that the involved entities do not change their behavior in response to a change of their reputation. As an example, the system may just provide a prospective buyer with indications of the probability that the seller will fail to deliver a

purchased item. This probability is the main property of the seller. It can change with time, but independently of the seller's reputation.

The other possibility is *sanctioning reputation systems*. The main assumption they make is that the involved entities are aware of the effect the reputation has on their benefits and thus adjust their behavior dynamically as their reputation changes. The main task of a reputation system in this case is to *sanction* misbehavior through providing correlation between the feedback the agent receives and the long-run profit she makes.

Cross-references

- ▶ Distributed Hash Table
- ▶ P2P Database
- ▶ Peer-to-Peer System
- ▶ Similarity and Ranking Operations
- ▶ Social Networks
- ▶ Trust in Blogosphere

Request Broker

Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA

Synonyms

Object request broker; Event broker; Storage broker

Definition

A Request Broker is a software manifestation of the Broker architectural pattern [3] that deals primarily with coordinating requests and responses, and managing resources among communicating entities in a distributed system. A Request Broker is usually found as part of middleware, which are layers of software that sit between applications, and the underlying operating systems, hardware and networks.

Historical Background

The mid to late 1980s established the TCP/IP protocol suite as the *de facto* standard suite of protocols for building networked applications. Contemporary operating systems provided a number of application programming interfaces (APIs) for network programming. The famous among these were the *socket* API that were tailored towards building TCP/IP-based applications. This era also saw the widespread use of killer TCP/IP-based applications, such as the File Transfer Protocol (FTP).

Despite the success of the socket API to build distributed applications, there were a number of challenges involved in developing these applications. First, the socket API was tedious to use and incurred a number of accidental complexities stemming from the use of type-unsafe C-language data types. Second, application programmers were responsible for handling the marshaling and demarshaling of the data types transferred between the communication entities, which involved a number of challenges. Notable among these were the need to address the byte ordering, and word size and padding issues arising from the heterogeneity in the hardware architectures. Third, server applications were able to provide only one primary functionality due to the lack of an object-oriented service abstraction. Finally, client applications had to explicitly bind to a location-dependent service, which made the applications inflexible and brittle.

Every distributed application that was developed had to reinvent the wheel and address these complexities. There was a compelling need to overcome these challenges by factoring out the commonly occurring patterns of network programming and provide them within reusable frameworks. This led to the notion of middleware [1,2], which provide reusable capabilities in one or more layers of software that sit between the application logic, and the operating systems, hardware and networks. A Request Broker is at the heart of such a middleware and performs a number of functions on behalf of the applications.

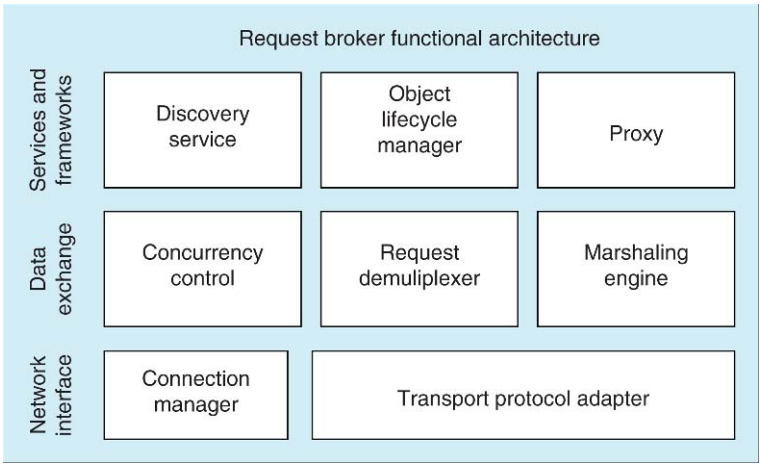
Request Brokering capabilities at the middleware layer started appearing with the advent of the Remote Procedure Call (RPC). Sun RPC was among the earlier middleware platforms that illustrated Request Brokers. Others that emerged thereafter included the Distributed Computing Environment (DCE), the Common Object Request Broker Architecture (CORBA), Java Remote Method Invocation (RMI), Distributed Component Object Model (DCOM) and .NET Remoting.

Foundations

A Request Broker implements the Broker architectural pattern [3]. The primary objectives of a request broker are to decouple clients and servers of a distributed application and provide reusable services, such as concurrency management, connection management, seamless transport-level networking support, data

marshaling, and location transparency. Figure 1 illustrates the commonly found functional blocks [5] in a Request Broker discussed below.

- *Proxy* – A Request Broker allows the separation of interface from implementation thereby decoupling the client of a service from the implementation of the service. A service is often described using interface definition languages to define the interfaces, the operations they support and data types that can be exchanged. Versioning of interfaces can also be provided within these descriptions. A proxy gives an illusion of the real implementation to the client.
- *Discovery services* – A Request Broker manages service discovery on behalf of clients. Some form of a service description, such as a URL or service name, is used by the broker to lookup a potentially remote implementation that offers the service. The broker will return to the client application a handle to the external service in the address space of the client so that the client can seamlessly invoke services on the handle via the Proxy. Depending on whether *pass by reference* or *pass by value* semantics are used, the execution of client requests may occur remotely on the server machine or locally in the client's address space, respectively.
- *Marshaling engine*– A Request Broker often specifies an encoding scheme or a serialization format for representing application data. Often there exist tools that read the interface descriptions of services and synthesize code that can marshal and unmarshal all the data types that are defined as serializable in the interface descriptions. This tool-generated code, which is usually called a *stub*, is linked into the fabric of the Request Broker, which manages data marshaling and unmarshaling on behalf of the application.
- *Concurrency control* – A server application may require finer-grained control on concurrency for scalability and performance. Request Brokers often provide sophisticated concurrency control mechanisms, such as *thread pools*, to handle application requests in a scalable and concurrent manner.
- *Object lifetime manager* – A server application may host multiple different services to optimally utilize resources. Often these services are provided in the form of objects implemented in a programming language. The Request Broker must manage several objects simultaneously in the system according to the policies dictating their lifetimes, e.g., transient or persistent. Other policies that tradeoff between memory footprint and performance include the *activate-on-demand* policy, which activates a service only on demand and for the duration of the request. This conserves resources but impacts performance.
- *Request demultiplexing and dispatching* – A broker may manage several hundreds of objects at a time. When requests arrive at a server, they must be efficiently demultiplexed and dispatched to the right object implementation.
- *Connection management and Transport Adapter* – Since applications are distributed they must communicate over different networking protocols. Request Brokers often enable applications to



Request Broker. Figure 1. Request broker functional architecture.

configure the choice of protocol to use for communication. For example, the unpredictable behavior of TCP/IP will not suffice for real-time applications in which case special transport protocols must be used. But the application must be shielded from these differences. Transport adapters [4] can provide these capabilities. For the transport protocol used, appropriate connection management capabilities are required. For example, connections may need to be purged periodically to conserve resources.

Apart from the functional architecture, a taxonomy of the capabilities provided by a Request Broker can be developed along the following orthogonal dimensions.

(1) *Communication models* – This dimension includes different classifications, such as (i) Remote Procedure Calls (RPC) versus Message Passing; (ii) Synchronous versus Asynchronous communications; (iii) Request/Response versus Anonymous Publish/Subscribe semantics; (iv) Client/Server versus Peer-to-Peer.
(2) *Location transparency* – This dimension includes mechanisms, such as object references or URLs, used by the Request Broker to hide the details of the service.
(3) *Type system support* – This dimension includes the richness of the data types that can be exchanged between the communicating entities, and the semantics of data exchange. For example, Request Brokers can support passing objects by value or by reference or both.
(4) *Interoperability and portability* – This dimension includes the degree of heterogeneity supported by the Request Brokers. For example, technologies such as CORBA are both platform- and language-independent, which makes them widely applicable but requires complex mapping between the platform- and language-independent representations to platform- and language-specific artifacts. Often this impacts the richness of data types that can be exchanged. On the other hand some technologies are language-dependent, e.g., Java RMI, or platform-dependent, e.g., DCOM and .NET Remoting.
(5) *Quality of service (QoS) support* – This dimension includes the capabilities provided by the Request Broker to support different QoS requirements of applications. Some brokers may be tailored to support real-time support while others are customized for persistence and transaction support.

Key Applications

Request Brokers are at the heart of distributed computing, and span a wide range of application domains including telecommunications, finance, healthcare, industrial automation, retail, grid computing, among others. Request Brokers with enabling technologies to support real-time applications have also been used to build distributed real-time and embedded systems found in domains, such as automotive control, avionics mission computing, shipboard computing, space mission computing, among others. Request Brokers are also found in systems that require management and scheduling of storage resources, or in large, event-based or content-management systems.

Future Directions

As applications become more complex, heterogeneous, and require multiple different and simultaneous quality of service properties, such as real-time, fault tolerance and security, the responsibilities of the Request Broker increase substantially. Brokering capabilities themselves will need to be distributed requiring coordination among the distributed brokers. Supporting multiple QoS properties will require design-time tradeoffs due to the mutually conflicting objectives of each QoS property. Run-time resource management will be a key in supporting the QoS properties since applications are increasingly demanding autonomic capabilities.

Experimental Results

Experimental research on Request Brokers is proceeding along the discussion articulated in the future trends. Service oriented computing is requiring Request Brokers to move beyond simple client-server or peer-to-peer computing to more advanced scenarios where the brokering capabilities are required across entire application workflows.

URL to Code

The following URLs provide more information on different Request Broker technologies and sample code.

CORBA is standardized by the Object Management group (http://www.omg.org).

Java RMI is a technology of Sun Microsystems (http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp).

DCOM and .NET Remoting are technologies from Microsoft (http://msdn2.microsoft.com/en-us/library/ms809340.aspx and http://msdn2.microsoft.com/en-us/library/2e7z38xb.aspx, respectively).

DCE is a technology standardized by the Open Group (http://www.opengroup.org/dce/).

Cross-references

- ▶ CORBA
- ▶ DCE
- ▶ DCOM
- ▶ .NET Remoting
- ▶ RMI
- ▶ Service Oriented Architecture
- ▶ Subsumed by Windows Communication Framework

Recommended Reading

1. Bakken D.E. Middleware. In Encyclopedia of Distributed Computing. J. Urban and P. Dasgupta (eds.). Kluwer, Dordrecht, 2001.
2. Bernstein P.A. Middleware: a model for distributed system services. Commun. ACM, 39(2):86–98, 1996.
3. Buschmann F., Meunier R., Rohnert H., Sommerlad P., and Stal M. Pattern-Oriented Software Architecture – A System of Patterns. Wiley, New York, 1996.
4. Gamma E., Helm R., Johnson R., and Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1995.
5. Schmidt D.C. Evaluating Architectures for Multi-threaded CORBA Object Request Brokers. Commun. ACM Special Issue on CORBA, 41(10):54–60, October 1998.

Residuated Lattice

Vilém Novák
University of Ostrava, Ostrava, Czech Republic

Synonyms

Structure of truth values

Definition

The residuated lattice is a basic algebraic structure accepted as a structure of *truth values* for fuzzy logic and fuzzy set theory. In general, it is an algebra

$$\langle L, \vee, \wedge, \otimes, \rightarrow, 0, 1\rangle$$

where L is a support, $\vee$ and $\wedge$ are binary lattice operations of join and meet, 0 is the smallest and 1 is the greatest element. The $\otimes$ is additional binary operation of *product* that is associative and commutative, and $a \otimes 1 = a$ holds for every $a \in L$. The $\rightarrow$ is a binary *residuation* operation that is adjoined with $\otimes$ as follows:

$$a \otimes b \leq c \quad \text{if and only if} \quad a \leq b \rightarrow c$$

for arbitrary elements $a, b, c \in L$. The residuation operation is a generalization of the classical implication.

Key Points

The residuated lattice is naturally ordered by the classical lattice ordering relation defined by

$$a \leq b \quad \text{if and only if} \quad a \wedge b = a.$$

In general, of course, there can exist incomparable elements in L. A typical property of the residuation operation is $a \rightarrow b = 1$ iff $a \leq b$. In words: $a \leq b$ iff the degree of implication $a \rightarrow b$ is equal to 1.

A typical example of residuated lattice is the *standard Łukasiewicz MV-algebra*

$$\langle [0,1], \max, \min, \otimes, \rightarrow, 0, 1\rangle$$

where $a \otimes b = \max\{0, a+b-1\}$ is Łukasiewicz product (conjunction) and $a \rightarrow b = \min\{1, 1-a+b\}$ is Łukasiewicz implication.

Another widely used residuated lattice is an MTL-algebra, where $L = [0,1]$, $\otimes$ as some left continuous *t-norm* (see TRIANGULAR NORMS) and $\rightarrow$ is the corresponding residuation.

Every boolean algebra is a residuated lattice. Thus, a special case of residuated lattice is also the boolean algebra for classical logic

$$\langle \{0,1\}, \vee, \wedge, \otimes, \rightarrow, 0, 1\rangle$$

where $\otimes = \wedge$ (i.e., $\otimes$ coincides with minimum) and $\rightarrow$ is the classical boolean (material) implication.

Both $\wedge$ as well as $\otimes$ are natural interpretations of logical conjunction. This means that there are two, in general different, conjunctions in fuzzy logic. In classical logic they coincide, though. The residuation $\rightarrow$ is natural interpretation of implication.

Negation is defined by

$$\neg a = a \rightarrow 0.$$

This operation coincides with classical negation in boolean algebra for classical logic, i.e., $\neg 0 = 1$ and $\neg 1 = 0$. In the standard Łukasiewicz MV-algebra (based on $[0,1]$) this operation reduces to $\neg a = 1 - a$.

The *biresiduation* operation is defined by

$$a \leftrightarrow b = (a \rightarrow b) \wedge (b \rightarrow a).$$

This operation coincides with with classical equivalence in boolean algebra for classical logic. In fuzzy logic, it is used as a natural interpretation of logical equivalence. In standard Łukasiewicz MV-algebra it gives $a \leftrightarrow b = 1 - |a - b|$, $a, b \in [0,1]$.

There are many more special kinds of residuated lattices. Except for boolean algebra, the most important are BL-algebra, Gödel algebra, product algebra and MV-algebra.

Cross-references

- ▶ Fuzzy Relation
- ▶ Fuzzy Set
- ▶ Triangular Norms

Recommended Reading

1. Esteva F. and Godo L. Monoidal t-norm based logic: towards a logic for left-continuous t-norms. Fuzzy Sets Syst., 124:271–288, 2001.
2. Hájek P. Metamathematics of Fuzzy Logic. Kluwer, Dordrecht, 1998.
3. Klement E.P., Mesiar R., and Pap E. Triangular Norms. Kluwer, Dordrecht, 2000.
4. Novák V. and Perfilieva I. Močkoř J. Mathematical Principles of Fuzzy Logic. Kluwer, Boston/Dordrecht, 1999.
5. Gottwald S. A Treatise on Many-Valued Logics. Research Studies, Baldock, Herfordshire, 2001.

Resource Allocation Problems in Spatial Databases

Donghui Zhang, Yang Du
Northeastern University, Boston, MA, USA

Synonyms

Facility-location problem

Definition

Assume that a franchise plans to open one or more branches in a state. How shall the locations of the new branches be allocated to maximally benefit the customers? Depending on whether some branches already exist and how to quantify the benefits to the customers, there are multiple forms of such resource allocation problems. In spatial databases, distance plays an important role. A customer is assumed to always visit the closest branch. Therefore it is beneficial to a customer if a new branch is opened at a location closer than her closest existing branch. The *max-inf optimal-location query* assumes the existence of a set of sites (already opened franchise branches), and aims to find a new location within a given area which benefits the largest number of customers. The *min-dist optimal-location query* also assumes the existence of a set of sites and aims to find a location for a new site which is optimal; but here the optimality is defined as minimization of the average distance from each customer to the nearest site. Compared with its max-inf counterpart, the min-dist optimal-location query takes into account the saved distance for each customer. The *k-medoid query* assumes that there does not exist any p as residential blocks) that minimize the average distance from every customer to the nearest picked location.

Historical Background

There exists an extensive literature in operations research on resource allocation problems, named the *facility location problems*. The most widely studied version is the *uncapacitated facility location (UFL)* problem.

Given a set of customers, a set of potential sites, a non-negative cost for opening each site, and a non-negative service cost between each site and a customer, the *UFL* problem is to find a subset of the potential sites so that the total cost (opening cost plus service cost) is minimum. Here, the term "uncapacitated" refers to the assumption that there is no limit on the number of customers that each site can serve.

The k-medoid problem is a variation of the UFL problem, where the set of customers and sites are identical, the opening cost of each site is zero, the service cost between a site and a customer is the distance between them, and there is an additional constraint that exactly k sites will be opened. The k-means problem is a related problem, where the potential sites can be anywhere.

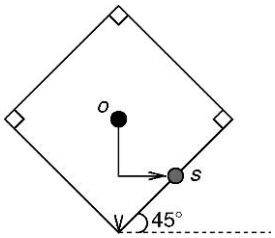
Facility location problems are (typically) NP-hard. Therefore research has focused on approximate algorithms with low computational complexity and small approximation errors. A recent survey of approximate algorithms for facility location problems appeared in [5]. Unlike spatial database research, the operations research assumes that all objects fit in memory and existing approaches typically scan through the dataset multiple times.

Foundations

When considering resource allocation problems in a large spatial database, the data usually reside in secondary storage and are indexed by spatial index structures so that scanning through all objects is avoided. The research goal is not to derive good asymptotic complexities, but to design efficient algorithms which, when applied to real datasets, incur a small number of I/O operations.

For both the max-inf and min-dist optimal-location queries under L_1 distance, there exist straightforward $O(n^2)$ solutions which find *exact* answers. Note that this differentiates the optimal-location queries from the NP-hard facility-location problem. However, such $O(n^2)$ algorithms are prohibitively slow in large spatial databases. Exploiting spatial index structures, [1] and [6] proposed solutions which are much faster (sub-linear in practice).

Similarly, the k-medoid query was examined in the context of spatial databases by utilizing spatial index structures. In particular, [2] relies on the clustering of higher-level index entries in an R-tree to avoid scanning through all objects.



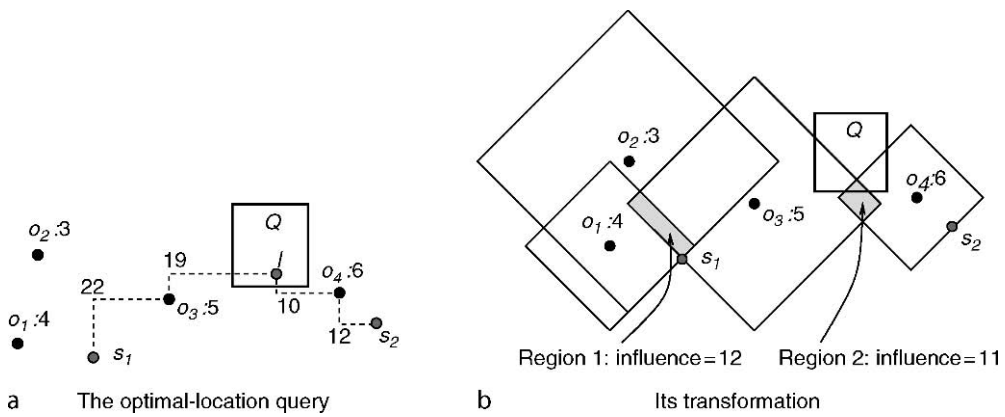
Resource Allocation Problems in Spatial Databases.
Figure 1. The *nn_buffer* of an object (or customer) under L_1 metric is a diamond (*rotated square*).

Max-Inf Optimal-Location Query

The max-inf optimal location (OL) query aims to find the location of a new site which benefits (or influences) the largest number of customers. Here the *influence* of a location is the number of customers which are closer to the location than to any existing site. To answer the max-inf OL query under L_1 metric, Du et al. [1] introduced the concept of *nn_buffer* and reduced the problem into finding a location with maximum overlap among *nn_buffers* of objects. Formally, the *nn_buffer* of an object is defined as follows.

Definition *Given an object o and its closest site s, the nn_buffer of o is a contour such that* $\forall l$ *on the contour,* $d(l, o) = d(o, s)$.

In other words, a location l is inside *o.nn_buffer* if and only if o is closer to l than to any site. As shown in Figure 1, the *nn_buffer* of an object o under L_1 metric is a diamond. The object o contributes to the influence of a location l, if and only if l is inside the *nn_buffer* of o. Furthermore, if the coordinates are rotated by 45° counter-clockwise, the *nn_buffers* become axis-parallel squares. Therefore, given a query region Q, an optimal location is a location l inside Q which maximizes the total weight of overlapped *nn_buffers*. Here the weight of an object tells how important an object is. For instance, if the object is an apartment complex, its weight can be the number of residents living in it. Figure 2 gives an example of the query with four objects and two sites and its corresponding



Resource Allocation Problems in Spatial Databases. Figure 2. In (a), l is an optimal location, with influence 11. The transformation in (b) shows that any location in the intersection between Q and region 2 is an optimal location.

transformation. Based on this observation, three solutions are proposed to answer the max-inf OL query under L_1 metric.

The first one is an R-tree based solution. It assumes that an R-tree is used to index the set O of objects. Furthermore, the R-tree is augmented with some extra information. Every object stores the L_1 distance to its closest existing site, and every index entry stores the maximum such distance of objects in the sub-tree. The solution follows two steps. The first step is to retrieve those objects from the R-tree whose *nn_buffers* intersect with Q. The objects are identified in increasing order of X coordinate in the rotated coordinate, even though the R-tree was built in the original coordinate. The second step is a plane-sweep process, which consumes the objects streamed in from the first step and computes the weight of overlapped *nn_buffers*. A naive plane-sweep solution has $O(n^2)$ cost, where n is the number of retrieved objects. However, it can be improved to $O(n \log n)$ by using a data structure called *aggregation SB-tree*. After processing all retrieved objects, the algorithm reports the location with maximum overlap as the answer.

The second solution is based on a specialized aggregation index called the *OL-tree*. The *OL-tree* is a balanced, disk-based, dynamically updateable index structure extended from the k-d-B-tree. This index is built in the rotated coordinate. Objects to be inserted into the OL-tree are *nn_buffers* versus points in the original k-d-B-tree. Local optimal location information in each subtree is stored along with the index entry referencing the sub-tree. Such information enables efficient processing algorithms that can answer the max-inf optimal-location query without examining all *nn_buffers* intersecting the query region Q. In the best case, the algorithm only needs to examine the root node of the tree.

These two solutions have interesting tradeoffs between storage cost and query efficiency. The R-tree based solution utilizes the existing R-tree structure. The storage cost is linear to the number of objects. However, as the objects are not pre-aggregated, a query needs to examine all objects whose *nn_buffers* intersect with the query range Q. If Q has large size, the query performance is poor. On the other hand, the OL-tree is a specialized aggregation index, whose space overhead is much higher, but provides faster query processing.

The third solution combines the benefits of the previous two approaches. As in the R-tree based solution, it uses an R-tree to store the objects. But to guide the search, it uses a small, in-memory OL-tree-like index structure. This index is named the *Virtual OL-tree (VOL-tree)*. It resembles the top levels of an OL-tree and it does not physically store any *nn_buffer*. A leaf entry plays a similar role to an index entry. It corresponds to a spatial range, and it logically references a node that stores (pieces of) *nn_buffers* in that range. If necessary, these *nn_buffers* can be retrieved from the R-tree dynamically. To prune the search space, the VOL-tree stores some (but not all) aggregated information of the *nn_buffers* in each level. This solution provides the best trade-off between space overhead and computational costs, as shown experimentally in [1].

Min-Dist Optimal-Location Query

The min-dist OL query aims to find the location of a new site which minimizes the average distance from each customer to the nearest site. Zhang et al. [6] proposed a progressive algorithm to find the exact answer of the Min-Dist OL query under L_1 metric. The algorithm works in two steps. In the first step, it limits the candidate locations to finite locations, which are the intersections of certain horizontal and vertical lines. The second step is a recursive partition-and-refine step. In this step, it partitions the query range Q into a few cells (by using some of the vertical and horizontal lines), and calculates $AD(\cdot)$ for the corners of these cells. Here $AD(l)$ is denoted as the average distance from each customer to the nearest site after opening a new site l. The smaller $AD(l)$ is, the better the new location l is. For each cell, the algorithm estimates the lower-bound of $AD(\cdot)$ among all locations in the cell and prunes the cell if its lower bound is larger than the minimum $AD(\cdot)$ already found. It repeatedly partitions the unpruned cells into smaller cells to further refine the result, until the actual optimal location is found.

To understand how the candidate locations are limited to finite locations, consider the example in Figure 3. The black dots are the objects and the thick-bordered rectangle is the query region Q. The shadowed region is composed of the horizontal extension of Q and the vertical extension of Q. The following theorem guarantees that the optimal location can be limited to finite candidates.

Theorem *Consider the set of horizontal (and vertical) lines that gothrough some object in the horizontal (and vertical) extension of Q or go through some corner of Q.*

The min-dist optimal location under L_1 metric can be found among the intersection points of these lines.

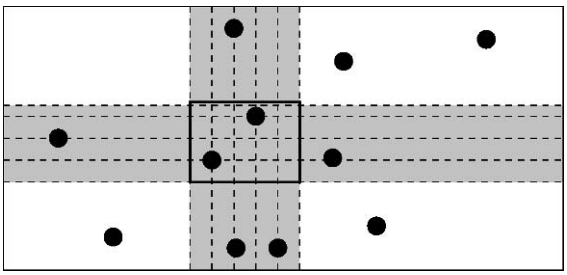
In order to prune some cells, the second step of the algorithm utilizes a novel lower-bound estimator for $AD(\cdot)$ of all locations in a cell. The following theorem describes how the estimator works.

Theorem *Let the corners of a cell C be* c_1, c_2, c_3, and c_4, *where* $\overline{c_1 c_4}$ *is a diagonal. Let the perimeter of C be p.*

$$\max\{\frac{AD(c_1)+AD(c_4)}{2}, \frac{AD(c_2)+AD(c_3)}{2}\} - \frac{p}{4}$$

is a lower bound of AD(l) for any location $l \in C$.

The progressive algorithm, which starts with one cell (the query region Q) and keeps partitioning it into smaller cells and trying to prune cells from the processing queue, has two advantages. First, it avoids computing $AD(\cdot)$ for all candidate locations, as all candidate locations inside a pruned cell are ignored. Second, it responds fast. An approximate answer is reported at the very beginning (after computing $AD(\cdot)$ for the four corners of Q), within a guaranteed error bound. As the algorithms runs, the candidate optimal location and the associated error rate are improved progressively, until the exact optimal location is found.



Resource Allocation Problems in Spatial Databases. Figure 3. The candidate locations are limited to the intersections of the dashed lines.

Disk-Based k-Medoid Query

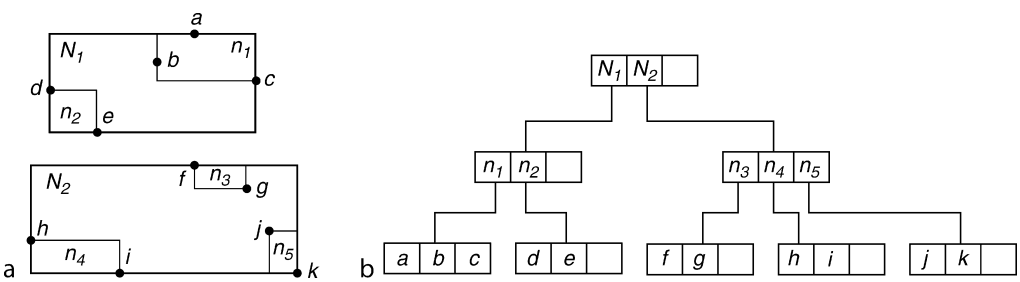
Mouratidis et al. [3] studied the k-medoid problem in large spatial databases. They assume that the data objects are spatial points indexed by an R-tree and the service cost between two points is defined by the distance between them. They propose the TPAQ (Tree-based PArtition Querying) algorithm that achieves low CPU and I/O cost. The TPAQ algorithm avoids reading the entire dataset by exploiting the grouping properties of the existing R-tree index.

Initially, TPAQ traverses the R-tree in a top-down manner, stopping at the topmost level that provides enough information for answering the given query. In the case of the k-medoid problem, TPAQ finds the topmost level with more than (or equal to) k entries. For instance, if $k = 3$ in the tree of Figure 4, TPAQ stops at level 1, which contains five entries, n_1 through n_5.

Next, TPAQ groups the entries of the partitioning level into k slots (i.e., groups.) To utilize the grouping properties of the R-tree index, TPAQ augments each retrieved entry n_i with a weight w and a center c. Here, the weight w is the number of points in the subtree referenced by n_i and the center c is the geometric centroid of the entry, assuming that the points in the sub-tree are uniformly distributed. To merge the initial entries into exactly k groups, TPAQ utilizes space-filling curves to select k seed entries which capture the distribution of points in the dataset. Then, each remaining entry is inserted into the slot whose weighted center is the closest to its center.

The final step of TPAQ is to pick k medoid objects, one from each group; TPAQ reports the weighted center of each group as the corresponding medoid.

Since the k-medoid problem is NP-hard, like any other practical algorithm TPAQ can only provide an approximate answer to the query. Experiments show that, compared to previous approaches, TPAQ achieves



Resource Allocation Problems in Spatial Databases. Figure 4. R-tree example.

comparable or better quality, at a small fraction of the cost (seconds as opposed to hours).

In [3], the authors also extended the above method to solve the medoid-aggregate query, where k is not known in advance. Given a user-specified parameter T, the medoid-aggregate query determines the smallest value of k and computes k-medoids such that the average distance from each object to the nearest medoid is smaller than T. The solution extends from TPAQ in two ways. First, without knowing k in advance, the R-tree top-down traversal stops at the level decided by the spatial extents and the expected cardinality of the entries. Second, multiple passes over the initial entries might be required to find the proper way of grouping the entries into slots.

Key Applications

Location-Based Services

Research on resource allocation problems is expected to help enhance the performance of location-based services based on the geographic proximity of clients to potential facilities of interest.

Spatial Decision Making

The efficient solutions to the resource allocation problems can provide valuable information to decision making. For example, to help provide candidate locations of a new branch.

Future Directions

One future research direction is to extend the solutions to other practical distance metrics. The existing solutions to both the max-inf and min-dist OL queries assume L_1 distance. Extending to L_2 (i.e., Euclidean) distance and road network distance is desirable. Similarly, extending the existing k-medoid solution to handle road network distance is interesting. Another future direction is to extend the queries to allow both the pre-existence of certain sites and the ability to find multiple locations. Existing solutions to the OL queries are limited to only one optimal location, and existing solutions to the k-medoid query are limited to zero existing site. Relaxing/avoiding these limitations will lead to more practical problems and solutions. The third future direction is to consider moving objects instead of static ones. In this case, the optimal locations and k-medoids should be continuously monitored over a set of moving objects [3,4].

Cross-references

▶ Nearest Neighbor Query
▶ Reverse Nearest Neighbor

Recommended Reading

1. Du Y., Zhang D., and Xia T. The optimal-location query. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 163–180.
2. Mouratidis K., Papadias D., and Papadimitriou S. Medoid queries in large spatial databases. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 55–72.
3. Papadopoulos S., Sacharidis D., and Mouratidis K. Continuous medoid queries over moving objects. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases, 2007, pp. 38–56.
4. U L.H., Mamoulis N., and Yiu M.L. Continuous monitoring of exclusive closest pairs. In Proc. 10th Int. Symp. Advances in Spatial and Temporal Databases, 2007, pp. 1–19.
5. Vygen J. Approximation algorithms for facility location problems (lecture notes). Technical Report, University of Bonn, Germany, 2005, pp. 1–59.
6. Zhang D., Du Y., Xia T., and Tao Y. Progressive computation of the min-dist optimal-location query. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 643–654.

Resource Description Framework

Michael Weiss
Carleton University, Ottawa, ON, Canada

Synonyms

RDF

Definition

RDF (Resource Description Framework) is a language for making statements about resources, i.e., for representing metadata [7]. Here, the term resource is intentionally used very broadly in the sense of any entity that can be uniquely identified by a URI (Universal Resource Identifier). This includes traditional web resources (such as web pages or images), as well as physical objects (such as devices) and humans about which the statements are made.

Historical Background

In 1998, Tim Berners-Lee described RDF in his vision for how metadata should be represented on the Semantic Web [4]. The first W3C Recommendation

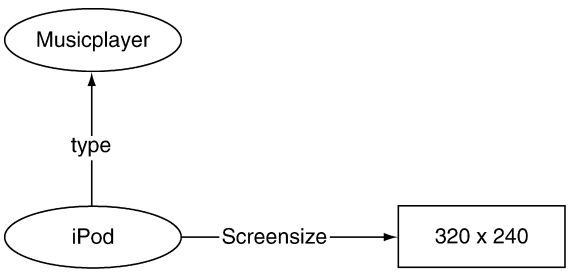
of RDF as a standard for encoding metadata was published in 1999. Several related efforts include the Meta Content Format (MCF) developed at Apple between 1995 and 1997, and the Dublin Core metadata standard [5] originally defined in 1995.

Foundations

Resource Description Framework

RDF is a core component of the layered specification of the semantic web (also known as the "layered cake"). It has its basis in the theory of semantic networks. The relationships between the elements of an RDF model can be presented in a number of ways: as a directed labeled graph (helpful to visualize the model), as object-attribute-value or resource-property-value triplets, or through an XML binding.

Figure 1 shows a simple example of an RDF graph. Oval nodes represent resources, and arrows between them represent properties. Property values that are themselves resources are shown as ovals, while literal values (such as strings) are shown as boxes. This figure makes two statements about an iPod music player: it is a type of music player, and it has a screen size of 320 × 240 pixels. The same information is shown as a set of resource-property-value triplets in Fig. 2. The resource `iPod` has two properties, `rdf:type` and `profile:ScreenSize`. The value of the property `rdf:type` is the type `MusicPlayer`, and the volume of the property `profile:ScreenSize` is the strip 320 × 240.



Resource Description Framework. Figure 1. Example of an RDF graph.

Resource	*Property*	*Value*
iPod	rdf:type	MusicPlayer
iPod	profile:ScreenSize	320x240

Resource Description Framework. Figure 2. Resource-property-value triplets.

Finally, Fig. 3 shows a corresponding XML binding. The XML binding starts with several namespace declarations, one for the RDF syntax, the other for the RDF schema that defines a vocabulary for describing mobile devices. This schema describes that a `MusicPlayer` is a kind of `MobileDevice`, and that `MobileDevice` resources have certain properties such as `ScreenSize`. The description of the `iPod` resource itself is contained in an `rdf:Description` element. It specifies a unique id through the `rdf:id` property, and a list of property values. As Fig. 2 illustrates, an `rdf:Description` element can contain multiple statements. However, it is not required that these statements are made within the same description. In fact, this allows a resource to be described in a decentralized manner. For example, the description of an iPod above could be extended – in a separate document – to make a statement about the iPod's owner without affecting the original description.

Key Applications

Another key application is FOAF (Friend of a Friend), a vocabulary for describing people and their relationships. A FOAF description can also contain information such as the organizations and projects a person works for and contributes to, documents they have created, and images that show them (Brickley & Miller, 2007). Information about a person can be distributed, leveraging the capability of RDF for decentralized representation. A person's

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:profile="http://www.examples.org/profile/mobile-device#">
  <rdf:Description rdf:ID="iPod">
    <rdf:type rdf:resource="#MusicPlayer"/>
    <profile:ScreenSize>320x240</profile:ScreenSize>
  </rdf:Description>
</rdf:RDF>
```

Resource Description Framework. Figure 3. XML binding for RDF.

own FOAF profile on their home page might only list the projects they work on and some of their friends. Other sites can provide additional information about that person using FOAF, for example, the page of a conference could list images of conference attendees. RDF has been applied to a variety of uses: annotating documents (e.g., Dublin Core [5]), representing device profiles (e.g., CC/PP [6]) and recently as exchange format in web services (e.g., MusicBrainz [9]). A popular use is in the RDF Site Summary (RSS) format, one of several RSS formats for lightweight content syndication in blogs [8].

Cross-references

- ▶ Dublin Core
- ▶ Metadata
- ▶ RDF Schema
- ▶ XML

Recommended Reading

1. Allemang D. and Hendler J. Semantic Web for the Working Ontologist: Modeling in RDF, RDFS and OWL. Morgan Kaufmann, 2008.
2. Berners-Lee T. What the semantic web can represent. Available online at: http://www.w3.org/DesignIssues/RDFnot.html, 1998.
3. Brickley D. and Guha R.V. RDF vocabulary description language: RDF schema. W3C recommendation. Available online at: http://www.w3.org/TR/rdf-schema/, 2004.
4. Brickley D. and Miller L. FOAF Vocabulary Specification. Available online at: http://xmlns.com/foaf/spec/, 2007.
5. Hillman D. Using dublin core. DCMI recommended resource. Available online at: http://dublincore.org/documents/usageguide/, 2005.
6. Klyne G., Reynolds F., Woordrow C., Ohto H., Hjelm J., Butler M., and Tran L. Composite capability/preference profiles (CC/PP): structure and vocabularies 1.0. W3C recommendation. Available online at: http://www.w3.org/TR/CCPP-struct-vocab/ 2004.
7. Manola F. and Miller E. RDF primer. W3C recommendation. Available online at: http://www.w3.org/TR/rdf-primer/, 2004.
8. RSS-DEV Working Group. RDF site summary (RSS) 1.0. Available online at: http://web.resource.org/rss/1.0/, 2000.
9. Swartz A. MusicBrainz: a semantic web service. IEEE Intell. Syst., 17(1):76–77, 2002.

Resource Description Framework (RDF) Schema (RDFS)

Vassilis Christophides
University of Crete, Heraklion, Greece

Synonyms

Conceptual schemas

Definition

An RDF schema (RDFS) is represented in the basic RDF model and provides (i) *abstraction* mechanisms, such (multiple) class or property subsumption and (multiple) classification of resources; (ii) *domain and range* class specifications to which properties can apply; (iii) *documentation facilities* for names defined in a schema.

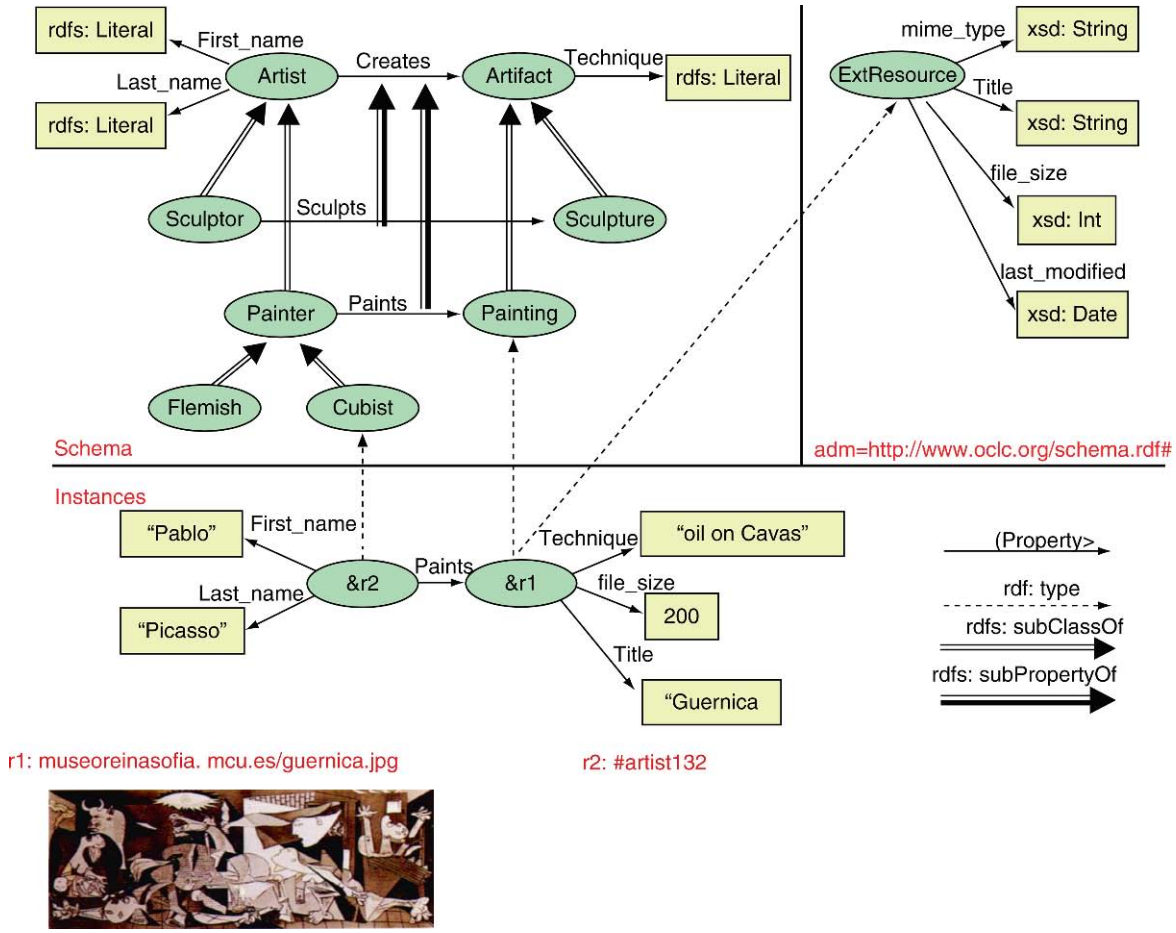
RDF/S follow the W3C design principles of interoperability, evolution and decentralization. In particular, it is possible to interconnect in an extensible way resource descriptions (by superimposing different statements using the same resource URIs) or schema namespaces (by reusing or refining existing class and property definitions) regardless of their physical location on the Web.

Key Points

Over the last decade, RDF and its accompanying RDFS specifications has been the subject of an extensive collaborative design effort. ([1]http://www.w3.org/RDF) RDF/S was originally developed as an application-neutral model to represent various kinds of descriptive information about Web resources, i.e., metadata. The W3C published the RDF Schema as Candidate Recommendation in 2000. Under the boost of the Semantic Web for transforming the Web into a universal medium for data, information, and knowledge exchange, refined versions of the RDF/S family of specifications have been published in 2004 [1–5]. As a matter of fact, RDF/S serves today as general purpose languages for consistent encoding, exchange and processing of available Web content, through a variety of syntax formats (XML or others). Their current design has been strongly influenced by recent advances in knowledge representation and aims to provide a simple *semantic layer* to the Web (no negation), as a base for more advanced query answering and reasoning services.

To present the core RDF/S modeling primitives, the example of a catalog of a cultural Portal (see Fig. 1) is used. To build this catalog, various cultural resources (e.g., Web pages of cultural sites) must be described from both a Portal and a Museum curator perspective. Figure 1 relies on an almost standard graphical notation, where nodes represent RDF/S resources (circles) or literals (rectangulars) while edges represent either user-defined (single arrows) or build-in (double and dashed arrows) properties.

The lower part of Fig. 1 depicts the description of an image (e.g., `museoreinasofia.mcu.es/guernica.`



Resource Description Framework (RDF) Schema (RDFS). Figure 1. A cultural Portal Catalog in RDF/S.

jpg) available on the Web. Hereforth the prefix "&" is used to abbreviate the involved resource URIs (e.g., &**r1**). In a first place, &**r1** is described from a Portal perspective as instance of the class named adm:ExtResource (uniqueness of names is ensured by using as prefix the corresponding schema URIs, like adm or cult in the example). More precisely, the statement (a triple in the RDF jargon) <&**r1**,rdf:type,adm:ExtResource> asserts that the resource &**r1** (subject) is of type (predicate) adm:ExtResource (object). Additionally &**r1** is stated to have two properties: one with name title and value the string "Guernica" (triple <&**r1**,adm:title,Guernica>) and, the other, with name file_size and value the integer 200 (triple < &**r1**, adm:file_size,200>). Resource &**r1** is also asserted as an instance of the class named Painting (triple <&**r1**,rdf:type,cult:Painting>) having a property technique with the literal value (string) "oil on canvas" (triple <&**r1**,cult:technique,oil on canvas>). &**r1** is further described by considering additional resources as for instance, &**r2** whose identifier (#artist132) is local to the Portal, i.e., not universally accessible in the Web. Resource &**r2** is then classified under Cubist and has a property first_name with value "Pablo" and a property last_name with value "Picasso." Finally, it is asserted that &**r2** and &**r1** are related thought the property paints (triple < &**r2**,cult:paints,&**r1**>).

The upper part of Fig. 1 depicts two RDF/S schemas (i.e., the namespace URIs **cult** = http://www.icom.com/schema.rdf# and **adm** = http://www.oclc.org/schema.rdf#) which define the classes and properties (i.e., the names) employed by the previous resource descriptions. In schema **cult**, the property creates(triple < cult:creates,rdf:type,rdf:Property>), is defined with domain the class Artist (triple < cult:creates,rdf:domain,cult:Artist>) and range the class Artifact (triple < cult:createsrdf:range,cult:Artifact>).

Note that properties are binary relations used to represent *attributes* of resources (e.g., `technique` with range a literal type (XML Schema datatypes could be used in this respect.) as well as *relationships* between resources (e.g., `creates` with range a class of resources). Furthermore, both classes and properties can be organized into *taxonomies* carrying inclusion semantics (multiple subsumption is also supported). For example, the class `Painter` subsumes `Cubist` (<`cult:Cubist, rdfs:subClassof, cult:Painter`>) while the property paints is subsumbed by creates (<`cult:paints,rdfs:subPropertyof,cult:creates`>). According to the RDFS semantics [3], rdfs:subClassof (or `rdfs:subPropertyof`) relations are transitive (and reflexive), thus enabling one to infer more triples (not depicted in Fig. 1.) than those explicitly stated (e.g., < `cult:Cubist, rdfs:subClassof,cult:Artist` > < &**r2**,rdf:type, cult:Painter > < &**r2**,rdf:type,cult:Artist > etc.).

To summarize, RDF/S properties are by default are *unordered* (e.g., there is no order between the properties `first_name` and `last_name`), *optional* (e.g., the property `mime-type` is not used) or with *multi-occurrences* (e.g., &**r2** may have two properties paints). Cardinality constraints for property domains/ranges (as well as inverse properties and Boolean class expressions) can be captured in more expressive ontology languages such as OWL. Furthermore, utility properties like `rdfs:label, rdfs:comment, rdfs:isDefinedBy` and `rdfs:seeAlso` are also available for documenting the development of a schema. Although not illustrated in Fig. 1, RDF/S also support structured values called *containers* for grouping statements, namely `rdf:Bag` (i.e., multi-sets) and `rdf:Sequence` (i.e., tuples), as well as, higher-order statements (i.e., *reified statements* whose subject or object can be another RDF statement). Figure 2 summarizes RDF/S axiomatic triples.

The RDF/S modeling primitives are reminiscent of knowledge representation languages (like Telos). Compared to traditional object or relational database models, RDF/S blurs the distinction between schema and instances. RDF/S schemas are *descriptive* (and not prescriptive designed by DB experts), *interleaved with the instances* (i.e., may cross abstraction layers when a resource is related through a property with a class) while may be *large* (compared to the size of instances). In particular, unlike objects (or tuples) RDF/S resources *are not strongly typed*:

```
rdf:type rdfs:domain rdfs:Resource .
rdfs:domain rdfs:domain rdf:Property .
rdfs:range rdfs:domain rdf:Property .
rdfs:subPropertyOf rdfs:domain rdf:Property .
rdfs:subClassOf rdfs:domain rdfs:Class .
rdf:subject rdfs:domain rdf:Statement .
rdf:predicate rdfs:domain rdf:Statement .
rdf:object rdfs:domain rdf:Statement .
rdfs:member rdfs:domain rdfs:Resource .
rdf:first rdfs:domain rdf:List .
rdf:rest rdfs:domain rdf:List .
rdfs:seeAlso rdfs:domain rdfs:Resource .
rdfs:isDefinedBy rdfs:domain rdfs:Resource .
rdfs:comment rdfs:domain rdfs:Resource .
rdfs:label rdfs:domain rdfs:Resource .
rdf:value rdfs:domain rdfs:Resource .

rdf:type rdfs:range rdfs:Class .
rdfs:domain rdfs:range rdfs:Class .
rdfs:range rdfs:range rdfs:Class .
rdfs:subPropertyOf rdfs:range rdf:Property .
rdfs:subClassOf rdfs:range rdfs:Class .
rdf:subject rdfs:range rdfs:Resource .
rdf:predicate rdfs:range rdfs:Resource .
rdf:object rdfs:range rdfs:Resource .
rdfs:member rdfs:range rdfs:Resource .
rdf:first rdfs:range rdfs:Resource .
rdf:rest rdfs:range rdf:List .
```

```
rdfs:seeAlso rdfs:range rdfs:Resource .
rdfs:isDefinedBy rdfs:range rdfs:Resource .
rdfs:comment rdfs:range rdfs:Literal .
rdfs:label rdfs:range rdfs:Literal .
rdf:value rdfs:range rdfs:Resource .

rdf:Alt rdfs:subClassOf rdfs:Container .
rdf:Bag rdfs:subClassOf rdfs:Container .
rdf:Seq rdfs:subClassOf rdfs:Container .
rdfs:ContainerMembershipProperty rdfs:subClassOf
rdf:Property .

rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .

rdf:XMLLiteral rdf:type rdfs:Datatype .
rdf:XMLLiteral rdfs:subClassOf rdfs:Literal .
rdfs:Datatype rdfs:subClassOf rdfs:Class .

rdf:_1 rdf:type rdfs:ContainerMembershipProperty .
rdf:_1 rdfs:domain rdfs:Resource .
rdf:_1 rdfs:range rdfs:Resource .
rdf:_2 rdf:type rdfs:ContainerMembershipProperty .
rdf:_2 rdfs:domain rdfs:Resource .
rdf:_2 rdfs:range rdfs:Resource .
...
```

Resource Description Framework (RDF) Schema (RDFS). Figure 2. RDF/S Axiomatic Triples.

- *RDF/S Classes do not define object or relation types*: an instance of a class is just a resource URI without any value/state (e.g., &**r1** is an instance of `Painting` regardless of any property associated to it);
- *RDF Resources may be instances of different classes* not necessarily pair wise related by subsumption: the instances of the same class may have associated quite different properties (e.g., see the properties of &**r1** which is multiply classified under the classes `ExtResource` and `Painting`);
- *RDF/S Properties are self-existent individuals* (i.e., decoupled from class definitions) which *may also be related through subsumption* (e.g., the property creates).

In addition, less rigid data models, such as those proposed for semi-structured databases, when they are not totally schemaless (such as OEM, UnQL), they cannot certainly exploit the RDF class (or property) subsumption taxonomies (as in the case of YAT). Finally, XML DTDs and Schemas have substantial differences from RDF schemas: (i) they cannot represent directed label *graphs* (Formally speaking, *RDF graph*s are not quite classical directed labeled graphs. First, a resource (e.g., paints) may occur both as a predicate (e.g., < &**r2**, cult:paints,&**r1**>) and a subject (e.g., < `cult:paints, rdf:domain, cult:Painter`>) of a triple. This compromises one of the more important aspects of graph theory: the intersection between the nodes and arcs labels must be empty. Second, in an RDF graph a predicate (e.g., `rdfs:subPropertyof`) may relate other predicates (<`cult:paints,rdfs:subPropertyof,cult:creates`>). Thus, the resulting structure is not a graph in the strict mathematical sense, because the set of arcs must be a subset of the Cartesian product of the set of nodes. There is an ongoing research on formalizing RDF using adequate graph models (e.g., bipartite graphs, directed hypergraphs).) (vs. rooted labeled *trees*); (ii) they cannot distinguish between *entity labels* (e.g., `Artist`) and *relationship labels* (e.g., `creates`); and (iii) they *constrain the structure* of XML documents, whereas an RDF/S schema simply *defines the vocabulary of class and property names* employed in RDF descriptions.

Cross-references

- ▶ Ontologies
- ▶ Resource Description Framework
- ▶ Semantic Web

Recommended Reading

1. Beckett D. RDF/XML syntax specification (revised). W3C recommendation. Available online at: http://www.w3.org/TR/rdf-syntax-grammar/, 2004.
2. Brickley D. and Guha R.V. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation. Available online at: http://www.w3.org/TR/rdf-schema/, 2004.
3. Hayes P. RDF Semantics. W3C Recommendation. Available online at: http://www.w3.org/TR/rdf-mt/, 2004.
4. Klyne G. and Carroll J. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation. Available online at: http://www.w3.org/TR/rdf-concepts/, 2004.
5. Manola F. and Miller E. RDF Primer W3C Recommendation. Available online at: http://www.w3.org/TR/rdf-primer/, 2004.

Resource Identifier

Greg Janée
University of California-Santa Barbara, Santa Barbara, CA, USA

Synonyms

Document identifier; UUID; GUID; Uniform resource identifier; URI

Definition

In a networked information system, a *resource identifier* is a compact surrogate for a resource that can be used to identify, retrieve, and otherwise operate on the resource. An identifier typically takes the form of a short textual string. An identifier must be *resolved* to yield the associated resource.

Key Points

Resource identifiers can be broadly characterized as either *locations*, which identify resources by where they reside, or *names*, which identify resources by properties intrinsic to the resources [2]. This distinction is not absolute, and identifiers can exhibit characteristics of both classes. Nevertheless, the distinction is useful in defining the relationship between identifiers and resources. Consider:

> *Can two distinct, yet identical resources have the same identifier?*
>
> *If a resource changes, must its identifier change?*

If the answer to these questions is yes, then the identifiers should be considered names; if no, locations. To take two well-known examples, International Standard

Book Numbers (ISBNs) are names, while HTTP URLs on the World Wide Web are locations.

Uniqueness

Uniqueness is the property that an identifier resolves to a single resource. The converse property – that every resource is identified by a single identifier, i.e., that identifier "aliasing" is avoided – is generally desirable, but is often not enforceable in systems that allow free generation of identifiers.

Broadly speaking, two approaches have been employed to guarantee uniqueness. The first is to incorporate into each identifier unique characteristics of the identified resource, for example a content-based signature, or characteristics of the context in which the resource and/or identifier system reside, for example a network address and timestamp. UUIDs incorporate both types of characteristics. The second approach is to acquire identifiers from an "authority" that maintains a centralized store of previously generated identifiers (identifier–resource associations are often stored as well). For scalability such systems are often arranged hierarchically so that a root authority, located at a well-known address, may delegate identifier generation and resolution requests to distributed sub-authorities. DNS and the Handle system are two well-known examples of this approach.

Persistence

Persistence is the property that an identifier continues to reference the associated resource over time. Strictly speaking, persistence is not a property of an identifier, or even a property at all; it's an outcome of the commitment of the operator of the identifier resolution system. A persistent identifier system is one that attempts to address known risks to persistence.

The risk of identifier breakage due to resource movement is universally mitigated by employing indirection: identifiers identify intermediate quantities which are maintained by resource owners to track current resource locations. In principle the indirection may be hidden from users, but for scalability reasons it is typically exposed. For example, the persistent uniform resource locator (PURL) system employs HTTP's redirection mechanism. The risk of breakage due to resource renaming has been mitigated in some systems by issuing so-called "semantics-free" identifiers; for example, DOIs are strings of digits with no external referent. However, the benefit of this approach must be balanced by the inscrutability of such identifiers to humans. Other notable persistent identifier systems include OpenURLs, which identify objects by metadata constraints, i.e., by intrinic resource properties; "robust hyperlinks," which append content-based signatures to locations, specifically URLs; and archival resource keys (ARKs), which incorporate a protocol for obtaining resource persistence guarantees and policies.

Other Properties

Additional desirable properties of resource identifiers include global scope, global uniqueness, extensibility, machine readability, recognizability in text, and human transcribability [3]. Identifiers that are subject to transcription errors may benefit from having error-correcting codes incorporated into them.

Cross-references

- ▶ Citation
- ▶ Digital Signatures
- ▶ Distributed Architecture
- ▶ Object Identity

Recommended Reading

1. Hilse H.-W. and Kothe J. (2006). Implementing persistent identifiers: overview of concepts, guidelines and recommendations. London/Amsterdam: Consortium of European Libraries and European Commission on Preservation and Access. http://nbn-resolving.de/urn:nbn:de:gbv:7-isbn-90-6984-508-3-8
2. Jacobs I. and Walsh N. (eds.) (2004). Architecture of the World Wide Web, Volume One. http://www.w3.org/TR/webarch/
3. Sollins K. and Masinter L. Functional Requirements for Uniform Resource Names. IETF RFC 1737, 1994. http://www.ietf.org/rfc/rfc1737.txt

Resource Scheduling

- ▶ Query Load Balancing in Parallel Database Systems

Restart Processing

- ▶ Crash Recovery

Restricted Data

- ▶ Statistical Disclosure Limitation For Data Access

Result Display

Catherine Plaisant
University of Maryland, College Park, MD, USA

Synonyms

Result display; Result overview; Preview; Data visualization

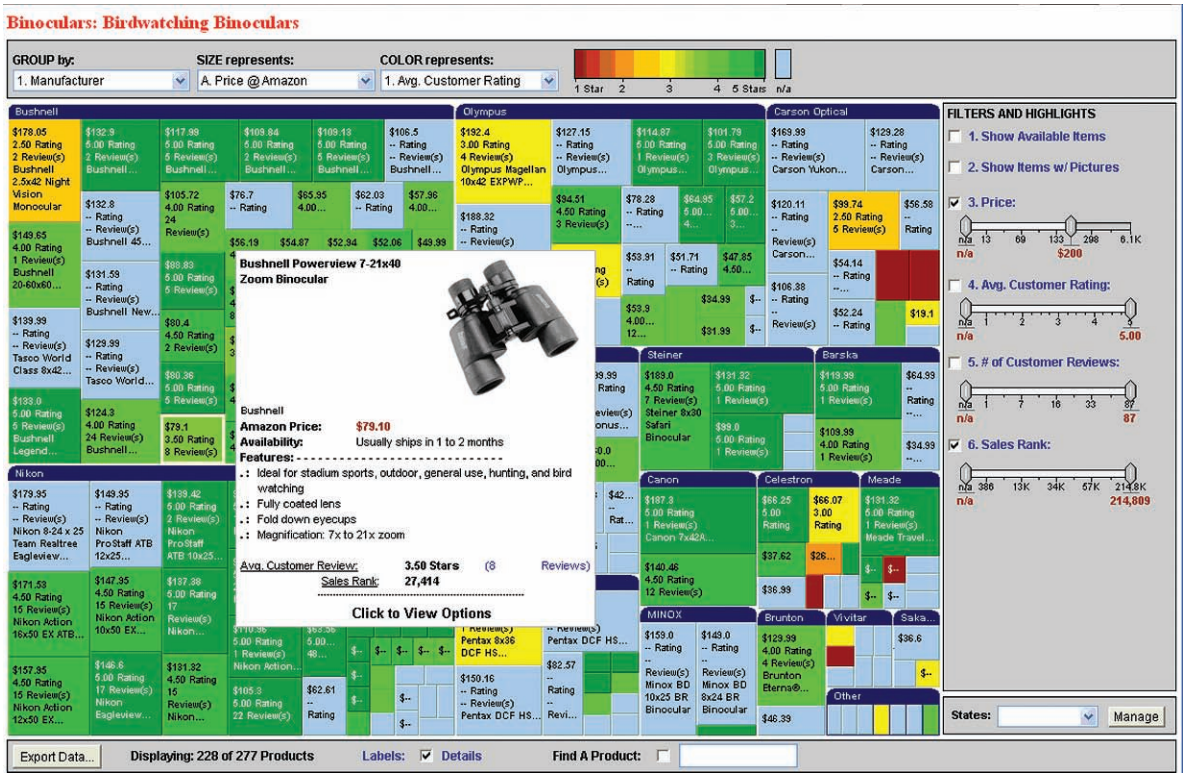
Definition

After formulating and initiating a search in a database, users review the results. The complexity of this task varies greatly depending on the users' needs, from selecting one or more top ranked items to conducting a complex analysis of the results in the hope of discovering an unknown phenomena. Displaying results includes providing and overview of the results and previews of items, manipulating visualizations, changing the sequencing of the results, adjusting the size of the results, clustering results by topic or attribute values, providing relevance feedback, examining individual items, and presenting explanatory messages such as restating the initial query.

Key Points

Displaying results is part of a dynamic and iterative decision-making process in which users initiate queries, review results and refine their queries. Users scan objects rapidly to determine whether to examine them more closely, or move on in the dataset. This process continues until the information need is satisfied, or the search is abandoned [3]. A visual result display typically includes interactive widgets to further filter the results, blurring the boundary between search and result display, e.g., dynamic queries entirely blend searching and result display [1].

The visual display of results relies on previews and overviews of the items returned by the search [2]. Graphical overviews indicate scope, size or structure and help gauge the relevance of items retrieved. Those



Result Display. Figure 1. After querying for birdwatching binoculars, users can review the results of their query using the Hive Group's treemap. Each box corresponds to a pair of binoculars and the size of the box is proportional to its price. Green boxes are best sellers, gray indicates unavailability. Results are grouped by manufacturer. Using the sliders on the right, users can filter results e.g., showing only items under $200.

overviews can vary from simple bar charts displaying the distribution of results over important attributes such as size or type, or consist of specialized visualizations such as interactive geographical maps, timelines, node link diagrams, conceptual topic maps etc. When no natural representation exist, more abstract overviews of the results can be used e.g., a treemap (Fig. 1). Multiple overviews are tightly coupled to facilitate synchronized browsing in multiple representations.

Previews consist of samples or summaries and help users select a subset of the results for detail review. Multimedia database require specialized preview mechanisms such as thumbnail browsers or video summaries allowing zooming and scene skipping. Both previews and overviews help users define more productive queries as they learn about the content of the database.

Users should be given control over what the size of the result set is, which fields are displayed, how results are sequenced (alphabetical, chronological, relevance ranked, and how results are clustered (by attribute value, by topics). One strategy involves automatic clustering and naming of the clusters for example in Vivisimo. Studies show that clustering according to more established and meaningful hierarchies such as the open directory might be effective. Translations may be proposed. Finally users need to gather information for decision making, therefore results need to be saved, annotated, sent by email or used as input to other programs such as visualization and statistical tools.

Cross-references

► Information Retrieval
► Video Querying
► Visualization

Recommended Reading

1. Ahlberg C. and Shneiderman B. Visual information seeking: tight coupling of dynamic query filters with starfield displays. In Proc. SIGCHI Conf. on Human Factors in Computing Systems, 1994, pp. 313–317.
2. Greene S., Marchionini G., Plaisant C., and Shneiderman B. Previews and overviews in digital libraries: designing surrogates to support visual Information-seeking, J. Am. Soc. Inf. Sci., 51 (3):380–393, 2000.
3. Shneiderman B. and Plaisant C. Designing the User Interface, (4th edn.). Addison-Wesley, 2005.

Result Overview

► Result Display

Result Ranking

► Web Search Relevance Ranking

Retrieval Models

► Web Search Relevance Ranking

Retrieval Models for Text Databases

► Structured Text Retrieval Models

Retrospective Event Processing

Opher Etzion
IBM Research Lab-Haifa, Haifa, Israel

Definition

Retrospective event processing is the detection of patterns on past events i.e., not done when the event occur, this can be done as part of existing event processing.

Historical Background

While the concept of "event pattern" typically refers to the events on-the-move, there are cases, in which it is required to use these patterns on past events, this typically happens in one of the following cases:

1. *Situation Reinforcement:* An event pattern designates the possibility that a business situation has occurred; in order to provide positive or negative reinforcement, as part of the on-line pattern detection, there is a need to find complementary pattern in order to assert or refute the occurrence of the situation.
2. *Retrospective Context.* In regular cases, contexts to start and look for patterns start with a certain event, and go forward until either some event occurs, or the context expires. There are cases in which the arrival of an event indicates the end of the context, however the context has not been monitored in the forward-looking way, and need to be monitored backwards.
3. *Patterns as queries.* Patterns are higher-level abstractions relative to SQL queries; in some cases

it is more convenient to use patterns, as higher-level languages on top of queries.

The issue of querying past information have been introduced in the area of *temporal databases* in which dealt with maintaining, querying and even updating past (and future) information. This included regular database queries or updates, however provided the infrastructure for storing past events.

The emerging of the event processing area have lead to the development of *event processing patterns* that initially was applied on the current events, retrospective event processing is the next logical step in getting the patterns on the past. It should be noted that there are three approaches to implementation of retrospective pattern language:

1. Extending pattern language to look at retrospective contexts
2. Adding patterns as an extension to SQL
3. Providing a hybrid language

Foundations

Taking the approach of extending event processing patterns for past information, this materializes in the following areas:

- *Retention policies:* in order to enable retrospective processing, event should be available beyond the original context of their processing, this leads to issues such as retention policies and vacuuming policies.
- *Grid storage:* one of the emerging areas in storing events and states for pre-determined term is storing events on in-memory stores on the grid.
- *Extending the notion of context:* the notion of temporal context should be extended to include past time intervals.
- *Automatic translation to SQL:* Assuming that the events are stored on a database, the pattern language should be translated to SQL. Again, it should be noted that an alternative approach is to include pattern extensions to SQL.

Key Applications

Use Cases for Situation Reinforcement

Anti-Money Laundering A person that has deposited (in aggregate) more than $20,000 within a single working day is a SUSPECT in money laundering. To reinforce the suspicion the following retrospective patterns are sought:

- There has been a period of week within the last year in which the same person has deposited (in aggregate) $50,000 or more and has withdrawn (in aggregate) at least $50,000 within the same week.
- The same person has already been a "suspect" according to this definition within the last 30 business days.

If any of these patterns are satisfied – the event "confirmed suspect" is derived.

The Greedy Seller Alert An electronic trade site provides the opportunity to customers to offer items for sale, but letting them conduct a bid, and provide bid management system (using a CEP system, of course). One of the services it provides to the customer is "alert on expensive sales":

If there have been at least two bidders, however, and none of them have matched the minimum price of the seller then this may be an indication of "too expensive bid."

To reinforce it, if at least two-thirds of the past bids of the same sellers have also resulted in a "too expensive bid" situation, then reinforce and send the seller a notification "you are too greedy."

Monitored Patient Alert A patient is hooked up to multiple monitors, the monitors are uncorrelated and each of them issues an alert when a certain threshold is passed (either up or down), results in most cases being false alarms. The physician can set up a "global monitoring system" which checks a pattern over recent time, e.g.,

- An alert has been given from the blood pressure monitor
- and Reinforcement condition:
  - If fever is more than 103F despite medication taken less than 2 hours ago, and blood pressure is strictly increasing in the last five measures then alert nurse.

Use Cases for Retrospective Contexts

Smart Retail The detected pattern is – "no item of a certain product reached the checkout in the last hour." This is a pattern that is detected hourly. If it is detected there is a retrospective context opened for this hour to check two retrospective patterns:

- If during that hour more than five customers took an item of that product from the shelf, but returned it after they have taken a competitive product (the information can be obtained by RFID tag on each item, an RFID reader for putting and removing items from the shopping cart).
- If no customer has taken an item of the product from the shelf, and did not take a competitor's product either.

Luggage Handling The reported event is – Luggage did not arrive.

Retrospective context – start at luggage check-in time.

- Collect all events related to this luggage (using the tag reading at various points).
- If no events found – notify the source airport to trace in their video tracking system.

Utilities Billing System Identified situation – customer has not paid 30 days after due date.

- Find out over the last billing cycle – has the customer addressed the customer center around issues with this bill, and obtain status.
- Look at the customer billings over the last year – determine maximal and average days of late in paying.
- Look at customers with the same Zip-code over the last billing cycle and determine the percentage of non-payment, late-payment (to determine if they are mail has significant delays in that area).

Use Cases for Patterns as Queries

Stock Trends Find all stocks that during the last month have satisfied the following conditions:

- The stock closing values at the end of the day were strictly increasing over a period of five consecutive working days, anywhere during this month.
- The stock value in the beginning at the end of the 5 days value was at least 30% more than its value at the beginning of the 5 days period.

Fraud Detection in On-Line Gaming Determine the case in which one commits identity theft, on the expense of the victim, consistently loose money to his partners in on-line poker game as a way of fraud (the example came from Oracle).

- Find a "consistent looser" – in a session that includes at least 30 poker games, in which a person loses in all the games, and the total loss is more than $20K.
- Find a person who has been a "consistent looser" at least twice, on distinct set of games.

Inventory Management The level of inventory is determined according to consumption prediction, to improve the inventory management, there are:

- Pre-planned orders that may be cancelled if the level of inventory does not justify order.
- Emergency orders that are requested if inventory goes below threshold in an unpredicted time.

The following patterns are requested:

- Find products for which "emergency order" has been requested at least twice during a period of the period of 30 days that ended in a date Last-Day (variable name) and no cancellation of pre-planned order during that period.
- Find products in which within 30 days there were cancellation of pre-planned order followed by an emergency order.
- Find products in the monitored period which all pre-planned orders have been cancelled, and no emergency orders occurred.

Cross-references

- ▶ Complex Event
- ▶ Context
- ▶ Event and Pattern Detection over Streams
- ▶ Event Pattern Detection
- ▶ Temporal Database

Recommended Reading

1. Arasu A., Babu S., and Widom J. The CQL continuous query language: semantic foundations and query execution. VLDB J., 15(2):121–142, 2006.
2. Deng M., Prasad Sistla A., and Wolfson O. Temporal conditions with retroactive and proactive updates. In Proc. 1st Int. Workshop on Active and Real-Time Database Syst., 1995, pp. 122–141.
3. Etzion O., Gal A., and Segev A. Retroactive and proactive database processing. In Proc. 4th Int. Workshop on Research Issues on Data Eng., 1994, pp. 126–131.
4. Gal A. and Etzion O. A multiagent update process in a database with temporal data dependencies and schema versioning. IEEE Trans. Knowl. Data Eng., 10(1):21–37, 1998.
5. Luckham D, The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, 2002.

6. Won J. and Elmasri R. Representing retroactive and proactive versions in bi-temporal databases. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 85–94.
7. Yang Y., Pierce T., and Carbonell J.G. A study of retrospective and on-line event detection. In Proc. 21st Annual Int. ACM SIGIR Conf. on Research and Development in Information Retrieval, 1998, pp. 28–36.

Reverse Nearest Neighbor Query

Dimitris Papadias[1], Yufei Tao[2]
[1]Hong Kong University of Science and Technology, Hong Kong, China
[2]Chinese University of Hong Kong, Hong Kong, China

Synonyms

Reverse nearest neighbor search; RNN query

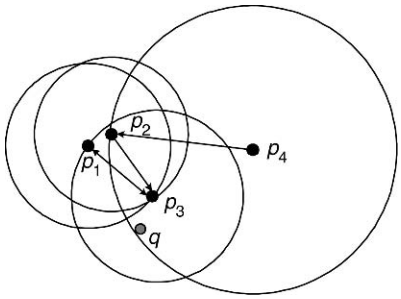
Definition

Given a multi-dimensional dataset P and a point q, a *reverse nearest neighbor* (RNN) query retrieves all the points $p \in P$ that have q as their nearest neighbor. The set RNN(q) of reverse nearest neighbors of q is called the *influence set* of q. Formally, RNN(q) = $\{p \in P \mid \neg \exists p' \in P$ such that $dist(p,p') < dist(p,q)\}$, where *dist* is a distance metric (Euclidean distance is assumed in the following examples).

The definition can also be extended to *reverse k nearest neighbors* (RkNN). Specifically, a RkNN query retrieves all the points $p \in P$ that have q as one of their k nearest neighbors. In this case, RkNN (q) = $\{p \in P \mid dist(p,q) \leq dist(p,p_k)$, where p_k is the k-th NN of $p\}$.

Historical Background

Reverse nearest neighbor queries were proposed in [4] and have received considerable attention due to their importance in several applications involving decision support, resource allocation, profile-based marketing, etc. Figure 1 shows an example R2NN query. The dataset P contains 4 points each associated with a circle covering its two nearest neighbors (NNs), e.g., the two NNs of p_4 (p_2, p_3) are in the circle centered at p_4. The result of a R2NN query q includes the "owners" of the circles that contain q, i.e., R2NN(q) = $\{p_3, p_4\}$. Let kNN(q) be the set of k nearest neighbors of q. Note that $p \in k$NN(q) does not necessarily imply $p \in$ RkNN(q), and vice versa. For instance, 2NN(q) = $\{p_1, p_3\}$, but p_1 does not belong to R2NN(q). On the other hand, although $p_4 \in$ R2NN(q), p_4 is not in 2NN(q).



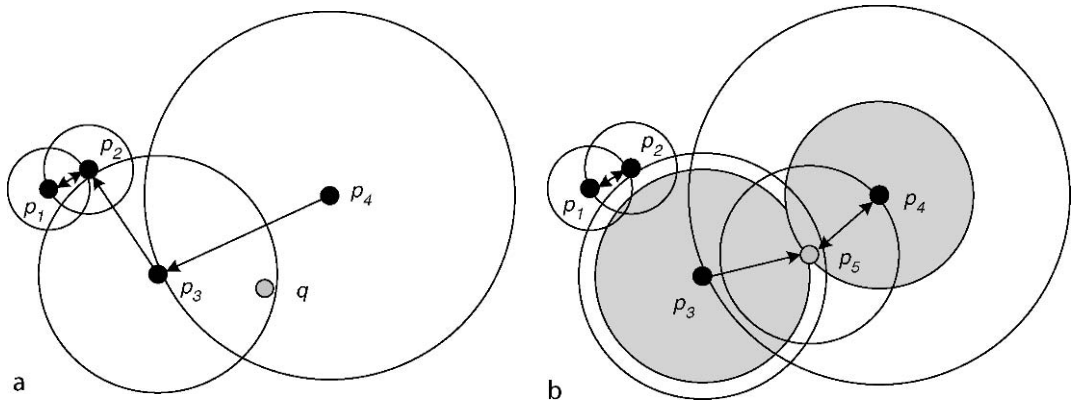
Reverse Nearest Neighbor Query. Figure 1. 2NN and R2NN examples.

Other versions of the problem include: (i) *continuous* RNN [1], where P contains linearly moving objects with fixed velocities, and the goal is to retrieve all RNNs of q for a future interval; (ii) *stream* RNN [5], where data arrive in the form of streams, and the goal is to report aggregate results over the RNNs of a set of query points, and (iii) *bichromatic* RNN [10] where, given two data sets P_1, P_2 and a query point $q \in P_1$, the goal is to find all the points $p_2 \in P_2$ that are closer to q than to any other object in P_1, i.e., $d(q, p_2) < d(p_1, p_2)$ for any $p_1 \in P_1$ and $p_1 \neq q$.

Foundations

Algorithms for RNN processing can be classified in two categories depending on whether they require pre-processing, or not. For simplicity, this section describes all methods for single RNN retrieval in 2D space, assuming that the dataset P contains points indexed by an *R-tree*. Their applicability to arbitrary values of k and dimensionality will be discussed at the end of the section.

The first RNN method, *KM* (Algorithms are referenced according to the author initials.) [4], pre-computes for each data point p its nearest neighbor NN(p). Then, it represents p as a *vicinity circle* (p, *dist*(p,NN(p))) centered at p with radius equal to the Euclidean distance between p and its NN. The MBRs of all circles are indexed by an *R-tree*, called the RNN-tree. Using the RNN-tree, the reverse nearest neighbors of q can be efficiently retrieved by a point location query, which returns all circles that contain q. Figure 2(a) illustrates *KM* using four data points, each associated with a vicinity circle. Since q falls in the circles of p_3 and p_4, the result of the query is RNN(q) = $\{p_3, p_4\}$. Because

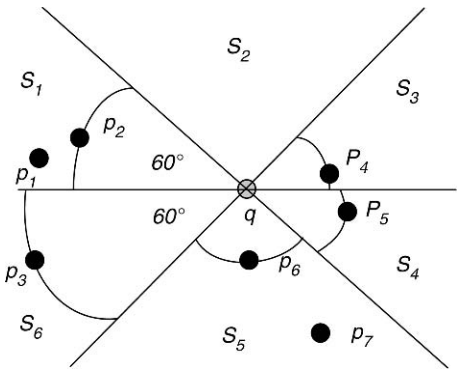


Reverse Nearest Neighbor Query. Figure 2. Illustration of *KM*.

the RNN-tree is optimized for RNN, but not NN search, *KM* uses an additional (conventional) *R-tree* on the data points for nearest neighbors and other spatial queries.

In order to avoid the maintenance of two separate structures, *YL* [13] combines the two indexes in the RdNN-tree. Similar to the RNN-tree, a leaf node of the RdNN-tree contains vicinity circles of data points. On the other hand, an intermediate node contains the MBR of the underlying points (not their vicinity circles), together with the maximum distance from every point in the subtree to its nearest neighbor. As shown in the experiments of [13], the RdNN-tree is efficient for both RNN and NN queries because, intuitively, it contains the same information as the RNN-tree and has the same structure (for node MBRs) as a conventional R-tree. *MVZ* [7] is also based on pre-computation. The methodology, however, is applicable only to 2D spaces and focuses on asymptotical worst case bounds (rather than experimental comparison with other approaches).

The problem of *KM*, *YL*, *MVZ*, and all techniques that rely on pre-processing, is that they cannot deal efficiently with updates. This is because each insertion or deletion may affect the vicinity circles of several points. Consider Fig. 2(b), where a new point p_5 needs to be inserted in the database. First, a RNN query is performed to find all objects (in this case p_3 and p_4) that have p_5 as their new nearest neighbors. Then, the vicinity circles of these objects are updated in the index. Finally, the update algorithm computes the NN of p_5 (i.e., p_4) and inserts the corresponding circle. Similarly, each deletion must update the vicinity circles of the affected objects. In order to alleviate the problem, Lin et al. [6] propose a method for bulk insertions in the RdNN-tree.

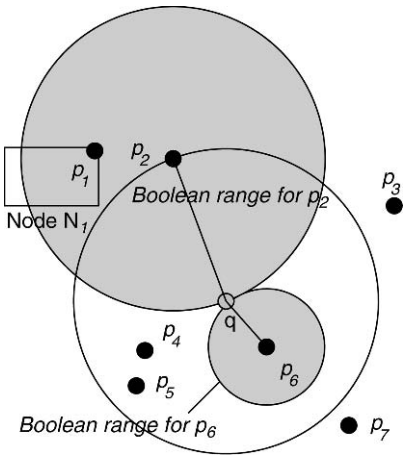


Reverse Nearest Neighbor Query. Figure 3. Illustration of *SAA*.

SAA [9] eliminates the need for pre-computing all NNs by utilizing some interesting properties of RNN retrieval. Consider Fig. 3, which divides the space around a query q into six equal regions S_1 to S_6. Let p be the NN of q in some region S_i; it can be proved that (i) either $p \in \text{RNN}(q)$ or (ii) there is no RNN of q in S_i. For instance, in Fig. 3 the NN of q in S_1 is point p_2. However, the NN of p_2 is p_1. Consequently, there is no RNN of q in S_1 and it is not necessary to search further in this region. The same is true for S_2 (no data points), S_3, S_4 (p_4, p_5 are NNs of each other) and S_6 (the NN of p_3 is p_1). The actual result is $\text{RNN}(q) = \{p_6\}$. Based on the above property, *SAA* adopts a two-step processing method. First, six constrained NN queries [2] retrieve the nearest neighbors of q in regions S_1 to S_6. These points constitute the *candidate* result. Then, at a second step, a *nearest neighbor query* is applied to find the NN p' of each candidate p.

If $dist(p,q) < dist(p,p')$, p belongs to the actual result; otherwise, it is a false hit and discarded.

The number of regions to be searched for candidate results increases exponentially with the dimensionality, rendering *SAA* inefficient even for three dimensions. *SFT* [8] follows a different approach that: (i) finds (using an R-tree) the K NNs of the query q, which constitute the initial candidates; (ii) it eliminates the points that are closer to some other candidate than q; (iii) it applies *boolean range queries* on the remaining candidates to determine the actual RNNs. Consider, for instance, the query of Fig. 4 assuming that K (a system parameter) is 4. *SFT* first retrieves the 4 NNs of q: p_6, p_4, p_5 and p_2. The second step discards p_4 and p_5 since they are closer to each other than q. The third step uses the circles $(p_2,dist(p_2,q))$ and $(p_6,dist(p_6,q))$ to perform two boolean ranges on the data R-tree. The difference with respect to conventional range queries is that a boolean range terminates immediately when (i) the first data point is found, or (ii) the entire side of a node MBR lies within the circle. For instance, N_1 contains at least a point within the range. Thus, p_2 is a false hit and *SFT* returns p_6 as the only RNN of q. The major shortcoming of the method is that it may incur false misses. In Fig. 4, although p_3 is a RNN of q, it does not belong to the 4 NNs of the query and will not be retrieved.

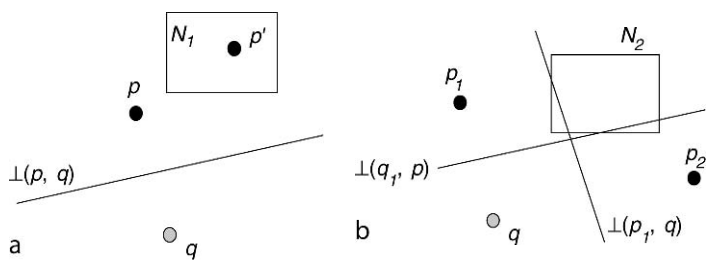


Reverse Nearest Neighbor Query. Figure 4. Illustration of *SFT*.

Similar to *SAA* and *SFT*, *TPL* [11] follows a filter-refinement framework. As opposed to *SAA* and *SFT* that require multiple queries for each step, the filtering and refinement processes are combined into a single traversal of the R-tree. In particular, *TPL* traverses the data R-tree and retrieves potential candidates in ascending order of their distance to the query point q because the RNNs are likely to be near q. Each candidate is used to prune node MBRs (data points) that cannot contain (be) candidates. For instance, consider the perpendicular bisector $\perp(p,q)$ between the query q and an arbitrary data point p as shown in Fig. 5(a). The bisector divides the data space into two half-planes: $PL_q(p,q)$ that contains q, and $PL_p(p,q)$ that contains p. Any point (e.g., p') in $PL_p(p,q)$ cannot be a RNN of q because it is closer to p than q. Similarly, a node MBR (e.g., N_1) that falls completely in $PL_p(p,q)$ cannot contain any candidate.

In some cases, the pruning of an MBR requires multiple half-planes. For example, in Fig. 5(b), although N_2 does not fall completely in $PL_{p_1}(p_1,q)$ or $PL_{p_2}(p_2,q)$, it can still be pruned since it lies entirely in the *union* of the two half-planes. In general, if p_1, p_2,...,p_{n_c} are candidate results, then any node whose MBR falls inside $\cup_{i=1\sim n_c} PL_{p_i}(p_i,q)$ cannot contain any RNN result. The filter step terminates, when there are no more candidates inside the remaining (i.e., non-pruned data space). Each pruned entry is inserted in a *refinement set* S_{rfn}. In the refinement step, the entries of S_{rfn} are used to eliminate false hits.

Table 1 summarizes the properties of each algorithm. Pre-computation methods cannot efficiently



Reverse Nearest Neighbor Query. Figure 5. Illustration of half-plane pruning in *TPL*.

Reverse Nearest Neighbor Query. Table 1. Summary of algorithm properties

	Dynamic data	Arbitrary dimensionality	Exact result	*Arbitrary k*
KM, YL	No	Yes	Yes	No
MVZ	No	No	Yes	No
SAA	Yes	No	Yes	Yes
SFT	Yes	Yes	No	Yes
TPL	Yes	Yes	Yes	Yes

handle updates. *MVZ* is suitable only to 2D spaces, while *SAA* is practically inapplicable for three or more dimensions. *SFT* incurs false misses, the number of which depends on the parameter K: a large value of K decreases the false misses but increases significantly the processing cost. Regarding the applicability of the existing algorithms to arbitrary values of k, precomputation methods only support a specific value (typically equal to 1), used to determine the vicinity circles. *SFT* can be adapted for retrieval of RkNN by setting a large value of K ($\gg k$) and replacing the boolean with *count* queries (that return the number of objects in the query range instead of their actual ids). *SAA* can be extended to arbitrary k as discussed in [11]. *TPL* can handle dynamic data for arbitrary values of k and dimensionality.

Key Applications

A number of applications for RNN can be found in [4] and [14]. Examples include:

Profile-Based Marketing

Assume that a real estate company keeps profiles of its customer set P based on their goals, i.e., each customer is a point in a vector space defined by the features of interest (e.g., house area, neighborhood etc). When a new estate q enters the market, a RNN query could retrieve the clients for which q constitutes the closest match to their interests.

Decision Support Systems

Consider that a franchise wants to open a new branch at location q so that it attracts a large number of customers from competitors based on proximity. This can be modeled as a bichromatic RNN query where P_1 corresponds to the competitor set and P_2 to the customer dataset. The result for a potential location q is the set of customers that are closer to q than any competitor.

Peer-to-Peer Systems

Assume that a new user q enters a P2P system. A RNN query retrieves among the existing users, the ones for which q will become their new NN based on the network latency. In a collaborative environment, q would inform such users about its arrival, so that they could address future requests directly to q, minimizing the network cost. Furthermore, the set RNN(q) reflects the potential workload of q; thus by knowing this set, each peer could manage/control its available resources.

Future Directions

Stanoi et al. [10] solve bichromatic RNN queries using *R-trees* to prune the search space. Benetis et al. [1] extend the *SAA* algorithm for continuous RNN queries. Yiu et al. [14] deal with reverse nearest neighbors in large graphs. Tao et al. [12] focus on RNN processing in *metric spaces*. Kang et al. [3] discuss the continuous evaluation of RNN queries in highly dynamic environments.

Experimental Results

Tao et al. [11] contains a comprehensive comparison of *SAA*, *SFT* and *TPL*. Each of the following references, except for [7], also contains an experimental evaluation of the proposed algorithm.

Data Sets

A common benchmark for RNN queries in the Euclidean space is the Tiger dataset: *http://www.census.gov/geo/www/tiger/*

In addition, the DBLP graph has been used for RNN queries in large graphs [14], while road networks have been applied in [12] and [11].

Cross-references

- ▶ Metric Space
- ▶ Nearest Neighbor Query
- ▶ R-Tree (and Family)

Recommended Reading

1. Benetis R., Jensen C., Karciauskas G., and Saltenis S. Nearest neighbor and reverse nearest neighbor queries for moving objects. VLDB J., 15(3): 229–250, 2006.

2. Ferhatosmanoglu H., Stanoi I., Agrawal D., and Abbadi A. Constrained nearest neighbor queries. In Proc. 7th Int. Symp. Advances in Spatial and Temporal Databases, 2001.
3. Kang J., Mokbel M., Shekhar S., Xia T., and Zhang D. Continuous evaluation of monochromatic and bichromatic reverse nearest neighbors. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 806–815.
4. Korn F. and Muthukrishnan S. Influence sets based on reverse nearest neighbor queries. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2000, pp. 201–212.
5. Korn F., Muthukrishnan S., and Srivastava D. Reverse nearest neighbor aggregates over data streams. In Proc. 28th Int. Conf. on Very Large Data Bases, 2002, pp. 814–825.
6. Lin K., Nolen M., and Yang C. Applying bulk insertion techniques for dynamic reverse nearest neighbor problems. In Proc. Int. Conf. on Database Eng. and Applications, 2003, pp. 290–297.
7. Maheshwari A., Vahrenhold J., and Zeh N. On reverse nearest neighbor queries. In Proc. Canadian Conf. Computational Geometry, 2002, pp. 128–132.
8. Singh A., Ferhatosmanoglu H., and Tosun A. High dimensional reverse nearest neighbor queries. In Proc. Int. Conf. on Information and Knowledge Management, 2003.
9. Stanoi I., Agrawal D., and Abbadi A., Reverse nearest neighbor queries for dynamic databases. In Proc. SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, 2000, pp. 44–53.
10. Stanoi I., Riedewald M., Agrawal D., and Abbadi A. Discovery of influence sets in frequently updated databases. In Proc. 27th Int. Conf. on Very Large Data Bases, 2001, pp. 99–108.
11. Tao Y., Papadias D., and Lian X. Reverse *k*NN search in arbitrary dimensionality. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 744–755.
12. Tao Y., Yiu M., and Mamoulis N. Reverse nearest neighbor search in metric spaces. IEEE Trans. Knowl. Data Eng., 18(9): 1239–1252, 2006.
13. Yang C. and Lin K. An index structure for efficient reverse nearest neighbor queries. In Proc. 17th Int. Conf. on Data Engineering, 2001, pp. 482–495.
14. Yiu M., Papadias D., Mamoulis N., and Tao Y. Reverse nearest neighbors in large graphs. IEEE Trans. Knowl. Data Eng., 18(4): 540–553, 2006.

Reverse Nearest Neighbor Search

► Reverse Nearest Neighbor Query

RF

► Relevance Feedback
► Relevance Feedback for Text Retrieval

Rich Media

► Video

Right-Time Data Warehousing

► Active and Real-Time Data Warehousing

Risk-Utility Tradeoff

► Statistical Disclosure Limitation For Data Access

Rewriting Queries using Views

Chen Li
University of California-Irvine, Irvine, CA, USA

Definition

Given a query on a database schema and a set of views over the same schema, the problem of query rewriting is to find a way to answer the query using only the answers to the views. Rewriting algorithms aim at finding such rewritings efficiently, dealing with possible limited query-answering capabilities on the views, and producing rewritings that are efficient to execute.

Historical Background

Query rewriting is one of the oldest problems in data management. Earlier studies focused on improving performance of query evaluation [9], since using materialized views can save the execution cost of a query. In 1995, Levy et al. [10] formally studied the problem and developed complexity results. The problem became increasingly more important due to new applications such as data integration, in which views are used widely to describe the semantics of the data at different sources and queries posed on the global schema. Many algorithms have been developed, including the bucket algorithm [11] and the inverse-rules algorithm [7,15]. See [8] for an excellent survey.

Foundations

Formally, a query Q_1 is *contained* in a query Q_2 if for each instance of their database, the answer to Q_1 is always a subset of that to Q_2. The queries are *equivalent* if they are contained in each other. Let T be a database schema, and $\mathcal{V}$ be a set of views on T. The *expansion* of

a query P using the views in $\mathcal{V}$, denoted by P^{exp}, is obtained from P by replacing all the views in P with their corresponding base relations. Given a query Q on T, a query P is called a *contained rewriting* of query Q using $\mathcal{V}$ if P uses only the views in $\mathcal{V}$, and P^{exp} is contained in Q as queries. P is called an *equivalent rewriting* of Q using $\mathcal{V}$ if P^{exp} and Q are equivalent as queries.

Examples: Consider a database with the following three relations about students, courses, and course enrollments:

```
Student(sid, name, dept);
Course(cid, title, quarter);
Take(sid, cid, grade).
```

Consider the following query on the database:

```
Query Q1: SELECT C.title, T.grade
FROM Student S, Take T, Course C
WHERE S.dept = 'ee' AND S.sid = T.sid AND
T.cid = C.cid;
```

The query asks for the titles of the courses taken by EE students and their grades. Queries and views are often written as conjunctive queries [4]. For instance, the above query can be rewritten as:

```
Q1(T, G) :- Student(S, N, ee), Take(S,
C, G), Course(C, T, Q).
```

Lower-case arguments (such as "ee") are used for constants, upper-case arguments (such as "T") for variables. The right-hand side of the symbol ":-" is the *body* of the query. It has three *subgoals*, each of which is an occurrence of a relation in the body. The constant "ee" in the first subgoal represents the selection condition. The variable S shared by the first two subgoals represents the join between the relations Student and Take on the student-id attribute. The variables T and G in the head of the query, which is the left-hand side of the symbol ":-", represent the final projected attributes.

Consider the following materialized views defined on the base tables:

```
Views: V1(S, N, D, C, G) :- Student(S, N,
D), Take(S, C, G);
 V2(S, C, T) :- Take(S, C, G), Course(C,
T, Q).
```

The SQL statement for the view definition of $V1$ is the following:

```
CREATE VIEW V1 AS
 SELECT S.sid, S.name, S.dept, T.cid,
T.grade
 FROM Student S, Take T
 WHERE S.sid = T.sid;
```

This view is the natural join of the relations Student and Take. Similarly, view $V2$ is the natural join of the relations Take and Course, except that the attributes about grades and quarters are dropped in the final results. The following is a rewriting of the query $Q1$ using the two views.

```
answer(T, G) :- V1(S, N, ee, C, G), V2(S,
C, T).
```

This rewriting takes a natural join of the two views on the attributes of student ids and course ids, then does a projection on the title and grade attributes. This rewriting can always compute the answer to the query on every instance of the base tables. In particular, after replacing each view in the rewriting with the body of its definition, the rewriting becomes the following expansion:

```
answer(T, G) :- Student(S, N, ee), Take
(S, C, G),
 Take(S, C, G'), Course(C, T, Q').
```

G' and Q' are fresh variables introduced during the replacements. This expansion is equivalent to the query, thus the rewriting is an equivalent rewriting of the query.

Now, assume in the definition of $V2$, there is another selection condition on the quarter attribute. The following is the view definition:

```
V2'(S, C, T) :- Take(S, C, G), Course(C,
T, fall2006).
```

That is, it only includes the information about the courses offered in the fall quarter of 2006. If only views $V1$ and $V2'$ are given, then the following is a rewriting of the query $Q1$:

```
answer(T, G) :- V1(S, N, ee, C, G),
V2'(S, C, T).
```

In particular, its expansion, which is obtained by replacing each view with the body of its definition, is the following:

```
answer(T, G) :- Student(S, N, ee), Take
(S, C, G),
 Take(S, C, G'), Course(C, T,
fall2006).
```

This expansion is contained in the original query, thus this rewriting is a contained rewriting of the query *Q*1. It is not an equivalent rewriting, since it does not include information about courses offered in other quarters. On the other hand, each fact in the answer to this rewriting is in the answer to the original query.

Suppose the view definition of *V*2 does not have the attribute about course ids. Then using this modified view and *V*1, there is no rewriting of the query, since the modified view does not have the course id to join with view *V*1. As another example, if the view definition of *V*1 does not keep the grade information, the following is the new view:

```
V1'(S, N, D, C) :- Student(S, N, D), Take
(S, C, G).
```

Using this new view and the original view *V*2, there is no rewriting to answer the query, since the views do not provide any information about grades, which is requested by the query. All these examples show that, when deciding how to answer a query using views, it is important to consider the conditions in the query and the views, including their selections, joins, and projections.

Algorithms. There are two classes of algorithms for rewriting queries using views: the first one includes the bucket algorithm [11] and its variants, and the second one includes the inverse-rules algorithm [7,15]. Notice that the number of possible rewritings of a query using views is exponential in the size of the query. Here the main idea of the bucket algorithm is explained using the running example, in which the query *Q*1 needs to be answered using the views *V*1 and *V*2. Its main idea is to reduce the search space of rewritings by considering each subgoal in the query separately, and deciding which views could be relevant to the query subgoal.

The bucket algorithm has two steps. In step 1, for each subgoal in the query, the algorithm considers each view definition, and checks if the body (definition) of the view also includes a subgoal that can be used to answer this query subgoal. For each view, if it includes a subgoal that can be unified with the query subgoal, and the query and the view are compatible after the unification, the corresponding head of the view definition is added to the bucket of this query subgoal. The following shows the buckets for the three query subgoals.

```
Student(S, N, ee): {V1(S, N, ee, C',
G')};
Take(S, C, G): {V1(S, N', D', C, G)};
Course(C, T, Q): {V2(S', C, T)}.
```

Each primed variable is a fresh variable introduced in the corresponding unification process. The bucket of the second query subgoal does not include the view *V*2 because the query subgoal requires the grade information be included in the answer, while the corresponding grade information in the view subgoal is not exported in the head of *V*2.

In step 2, the algorithm selects one view from each bucket, and combines the views from these buckets to construct a contained rewriting. The following is a contained rewriting:

```
Q1(T, G) :- V1(S, N, ee, C', G'), V1(S,
N', D', C, G), V2(S', C, T).
```

The final output of the algorithm is the union of contained rewritings in order to maximize the set of answers to the query using the views, since these rewritings could produce different pieces of information.

One main advantage of the bucket algorithm is that it can prune those views that do not contribute to a condition in the query, thus it can reduce the number of candidate rewritings to be considered. One limitation of the algorithm is that each query subgoal introduces a view in a rewriting. For instance, in the example above, view *V*1 could be used to answer the first two query subgoals. But the algorithm needs to use three view instances in each candidate rewriting, which requires more postprocessing steps to simplify this rewriting. In addition, the algorithm does not use the fact that if a view can be used to cover a query subgoal using a view variable that is not exported in the head of the view, then the view has to cover all the query subgoals that use the corresponding query variable. Based on these observations, a new algorithm, called MiniCon, was developed to make the rewriting process significantly more efficient [14]. A similar idea was used in the shared-bucket-variable (SVB) algorithm [13].

In some cases, especially in the context of data integration, where a view is a description of the content at a data source, the views could have limited query capabilities. For instance, imagine the case where the view *V*1 above is a materialized table, such that it can be accessed only if a student id is provided to the table, and the table can return its information about that student id. The table does not accept arbitrary queries such as "return all records," or "retrieve all information about students from the CS department." These limitations on the views present new challenges for the development of query-rewriting algorithms. The problem in this setting was studied in [16]. It is shown that

the the inverse-rules algorithm [7] can handle such restrictions with minor modifications.

Other algorithms have been developed to study variants of the query-rewriting problem. The CoreCover algorithm [2] was developed for the problem of generating an *efficient* equivalent rewriting efficiently. There was also a study [1] for the case where the query and the views can have comparison conditions such as `salary > 30K` and `year <= 2004`. The work in [6] studied how to compute a set of views with a minimal size to compute the answers to a set of queries. In some settings, applications need to find a rewriting called "maximally contained rewriting," which can compute the maximal set of answers to the query using the views. The problem is also different depending on whether the closed-world assumption is taken (as in data warehousing, in which each materialized view is assumed to include all the facts satisfying the view definition) or the open-word assumption is taken (as in data integration, in which each view includes a subset of the facts satisfying the view definition). In the literature there is another related problem called "query answering." See [3] for a comparison between "query rewriting" and "query answering."

Key Applications

The problem of rewriting queries using views is related to many data-management applications, including information integration [12,18], data warehousing [17], and query optimization [5].

Cross-references

- ▶ Answering Queries Using Views
- ▶ Closed-World Assumption (CWA)
- ▶ Data Integration
- ▶ Data Warehouse
- ▶ Global-as-View (GAV)
- ▶ Local-as-Views (LAV)
- ▶ Open-World Assumption (OWA)
- ▶ Query Containment
- ▶ Query Optimization

Recommended Reading

1. Afrati F.N., Li C., and Mitra P. Answering Queries Using Views with Arithmetic Comparisons. In Proc. 21st ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2002, pp. 209–220.
2. Afrati F., Li C., and Ullman J.D. Generating Efficient Plans Using Views. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2001, pp. 319–330.
3. Calvanese D., Giacomo G.D., Lenzerini M., and Vardi M.Y. View-Based Query Processing: On the Relationship Between Rewriting, Answering and Losslessness. In Proc. 10th Int. Conf. on Database Theory, 2005, pp. 321–336.
4. Chandra A.K. and Merlin P.M. Optimal Implementation of Conjunctive Queries in Relational Data Bases. In Proc. 9th Annual ACM Symp. on Theory of Computing, 1977, pp. 77–90.
5. Chaudhuri S., Krishnamurthy R., Potamianos S., and Shim K. Optimizing Queries with Materialized Views. In Proc. 11th Int. Conf. on Data Engineering, 1995, pp. 190–200.
6. Chirkova R. and Li C. Materializing views with minimal size to answer queries. In Proc. 22nd ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 2003, pp. 38–48.
7. Duschka O.M. and Genesereth M.R. Answering Recursive Queries Using Views. In Proc. ACM SIGACT-SIGOPS 16th Symp. on the Principles of Dist. Comp., 1997, pp. 109–116.
8. Halevy A.Y. Answering queries using views: A survey. VLDB J., 10(4):270–294, 2001.
9. Larson P.Å. and Yang H.Z. Computing Queries from Derived Relations. In Proc. 11th Int. Conf. on Very Large Data Bases, 1985, pp. 259–269.
10. Levy A., Mendelzon A.O., Sagiv Y., and Srivastava D. Answering Queries Using Views. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 1995, pp. 95–104.
11. Levy A., Rajaraman A., and Ordille J.J. Querying Heterogeneous Information Sources Using Source Descriptions. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 251–262.
12. Li C. Query Processing and Optimization in Information-Integration Systems. Ph.D. Thesis, Computer Science Dept., Stanford Univ., 2001.
13. Mitra P. An algorithm for answering queries efficiently using views. In Proc. the 12th Australasian Database Conf. 2001, pp. 99–106.
14. Pottinger R. and Levy A. A Scalable Algorithm for Answering Queries Using Views. In Proc. 26th Int. Conf. on Very Large Data Bases, 2000.
15. Qian X. Query folding. In Proc. 12th Int. Conf. on Data Engineering, 1996, pp. 48–55.
16. Rajaraman A., Sagiv Y., and Ullman J.D. Answering Queries Using Templates with Binding Patterns. In Proc. 14th ACM SIGACT-SIGMOD-SIGART Symp. Principles of Database Systems, 1995, pp. 105–112.
17. Theodoratos D. and Sellis T. Data warehouse configuration. In Proc. 23th Int. Conf. on Very Large Data Bases, 1997, pp. 126–135.
18. Ullman J.D. Information Integration Using Logical Views. In Proc. 6th Int. Conf. on Database Theory, 1997, pp. 19–40.

RMI

Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA

Synonyms

Remote method invocation

Definition

Java Remote Method Invocation (RMI) [1,2] is a Java language-based technology to achieve distributed computing among distributed Java virtual machines.

Key Points

Java RMI is a Java language-dependent technology for distributed computing. It provides seamless distributed communication between Java virtual machines. RMI uses object serialization and offers true object-oriented polymorphism even across distributed address spaces. Since RMI is based on Java, it brings the power of safety, concurrency and portability to distributed applications. In developing their applications, programmers must explicitly indicate which interfaces will be available as a remote service by extending the java.rmi.Remote interface.

A special quality of RMI is its ability to dynamically load new objects into an address space. For example, if a remote service has undergone change and extended its capabilities, it is feasible for RMI to dynamically load the new class in the client's address space.

Another attractive feature of RMI is its ability to allow entire behaviors of objects to be sent to remote entities. At the remote end, it is then feasible to activate a local copy of the passed object with its behavior. These techniques are useful in load balancing and faster response times.

RMI can allow clients behind firewalls to contact remote servers. This capability enables clients to reside within applets. RMI also provides interoperability with other broker technologies, such as CORBA, by supporting RMI over CORBA IIOP.

Cross-references

- ▶ Client-Server Architecture
- ▶ CORBA
- ▶ DCE
- ▶ DCOM
- ▶ J2EE
- ▶ .NET Remoting
- ▶ Request Broker
- ▶ SOAP

Recommended Reading

1. Sun Microsystems. Java Remote Method Invocation. 1996.
2. Sun Developer Network. Remote Method Invocation. Available at: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp.

RNN Query

▶ Reverse Nearest Neighbor Query

Road Network Databases

▶ Road Networks

Road Networks

Cyrus Shahabi
University of Southern California, Los Angeles, CA, USA

Synonyms

Spatial network databases; Road vector data; Road network databases

Definition

In *vector* space, the distance between two objects can be computed as a function of the components of the vectors representing the objects. A typical distance function for multidimensional vector spaces is the well-known Minkowski metric, with the Euclidean metric as a popular case for two-dimensional space. Therefore, the distance computation in multidimensional vector spaces is fast because its complexity depends on the number of dimensions which is limited to two or three in geospatial applications. However, with *road-networks*, the distance between two objects is measured by their *network distance*, i.e., the length of the shortest path through the network edges that connects the two locations. This computationally expensive network-based metric mainly depends on the connectivity of the network. For example, representing a road network as a graph with e weighted edges and v vertices, the complexity of the Dijkstra algorithm to find the minimum weighted path between two vertices is $O(e + vLogv)$. Therefore, several distance-based queries, such as nearest-neighbor queries, cannot be performed efficiently in road-networks. One option is to estimate the distance between two objects by their Euclidean distance. Unfortunately, as shown in [13], the Euclidean distance is not a good approximation of the network distance with real-world

road-networks. Therefore, in the past several years, many studies have been investigating new techniques to index road-networks and/or pre-compute distances in order to expedite distance-based query processing on road-networks.

Historical Background

The main challenge with query processing in road-networks is the high complexity of network distance computation. This is important since many applications, especially those dealing with moving objects, require frequent, fast and on-the-fly computation of distances between a query point and several points of interests. The earliest work that studied this challenge of fast distance computation in road-network databases is by Shahabi et al. [13]. In this paper, the authors proposed an embedding technique to transform the road network to a high dimensional space in which fast Minkowski metrics can be utilized for distance measurement. The results are still approximation of the actual distances but much more accurate than using Euclidean distances on the points' geographical coordinates.

Besides this transformation-based approach to perform fast network distance computation, three other main approaches are based on either indexing the network or pre-computing the network distances or a hybrid of the two. A prominent work in the indexing category is by Papadias et al. [10], which proposes an architecture for road-networks that uses a disk-based network representation. Their approach is based on the fact that the current algorithms (e.g., Dijkstra) for computing the distance between a query object q and an object O in a network will automatically result in the computation of the distance between q and the objects that are (relatively) closer to q than O. This approach applies an optimized network expansion algorithm with the advantage that the network expansion only explores the objects that are closer to q and computes their distances to q during expansion. The advantages of this approach are: (i) it offers a method that finds the exact distance in networks, and (ii) the architecture can support other spatial queries like range search and closest pairs. There are other studies that similar to this work try to combine traditional spatial access methods with some sort of network representation and expansion method such as [3,5,7]. The main disadvantage of these network-expansion approaches is that they perform poorly when the objects are not densely distributed in the network because then they require to retrieve a large portion of the network for distance computation. A rather different indexing approach is proposed in [4], termed distance signature. This approach categorizes the distances between objects and network nodes into groups and then encodes these groups.

A representative work for pre-computation approaches is the work by Sankaranarayanan et al. [12], which proposes a framework called SILC for computing the shortest distance between vertices on a spatial network. The proposed framework pre-computes the shortest path between all pairs of vertices, which in turn results in fast distance computation. Examples of other studies that used some sort of pre-computation to expedite network distance computation are [6,2].

A hybrid approach that combines an indexing technique with pre-computation is proposed in [8]. This approach is based on partitioning and then indexing a large network to small *network Voronoi* regions, and then pre-computing distances both within and across the regions.

There are also many variations in the applications and query types on road-networks that require special-purpose index structures and/or query processing algorithms for efficient network distance computation. Some of these applications and query-types are reviewed below.

As mentioned, the fast computation of network distance becomes more critical when objects are moving. Most of the aforementioned techniques can indeed be used for querying points of interests from a moving object. However, some optimization can be performed if one wants to update the query results as the query point moves. One main representative query type here is the Continuous k-nearest-neighbor (C-kNN) queries. C-kNN on road-networks maintains the k nearest neighbors in network distance as the query object moves on the road network, which has its own unique challenges as opposed to C-kNN in Euclidean spaces. Sample studies focusing on moving objects in road-networks are [1,11].

Other novel applications include [9], which tries to address NN queries by proposing a novel travel time network that integrates both spatial networks and real-time traffic event information. In [15], the authors propose and solve Aggregate nearest neighbor queries (ANN) in the context of large road networks. In [14], the authors study the novel problem of optimal

sequenced route (OSR) query in both vector and road-network spaces. The OSR query tries to find a route of minimum length starting from a given source location and passing through a number of typed locations in a specific sequence imposed on the types of the locations. The paper proposes a pre-computation approach to OSR query by exploiting the geometric properties of the solution space and relating it to additively weighted Voronoi diagrams.

Foundations

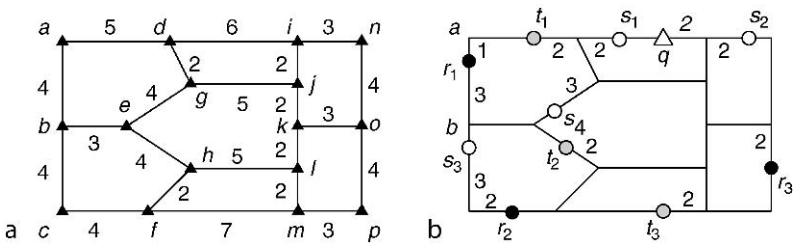
A popular class of queries in geospatial applications is the class of distance-based queries. To answer these queries, the distance between one or more query points or regions with some points or areas of interests must be computed. A frequently used member of this class is the k nearest neighbor (kNN) query where the k closest points to a query point are requested. For example, many car navigation systems provide the feature to ask for the k closest gas stations to the vehicle's current location. One way to answer this query is to estimate the distance by computing the Euclidean distance between the vehicle's geographical coordinates (i.e., latitude and longitude) and all the gas stations' coordinates in the vicinity. The problem with this approach is that the Euclidean distance corresponds to the *air-distance* between the vehicle and the gas station. Basically, if the car could have flown, then the Euclidean distance would have been the accurate distance between the car and the gas station. Unfortunately, most cars are restricted to the underlying road network. Hence, the actual distance between the car and the gas station depends on the connectivity of the underlying network and usually is very different than the Euclidean distance. In [13], it is shown that in real-world road-networks, Euclidean distance does not yield a good approximation of the network distance.

Formal Definition of the Problem

The challenge with computing network-distance is that the complexity of its computation depends on the number of vertices and edges of the underlying network, which is normally very large for real road-networks. Now imagine computing this distance continuously from a moving vehicle to several gas stations and one would appreciate the efforts in developing new techniques to compute this distance as fast as possible. To formally define this problem, first a formal model to represent road networks is provided and then using this model, the concept of network-distance is defined more accurately.

A road-network can be modeled as a weighted graph. Consider the weighted undirected (Many spatial networks consist of directed edges and hence must be modeled as directed graphs. Throughout this entry, undirected graphs are used for simplicity.) graph $G = (V, E)$ as the two sets V of vertices, and $E \subseteq V \times V$ of edges. Each edge of E, directly connecting vertices u and v, is represented as the pair $[u, v]$. Each vertex v represents a 2-d point $(v.x, v.y)$ in a geometric space (e.g., an intersection in a road network). Hence, each edge is also a line segment in that space (e.g., a road segment). A numeric weight (cost) w_{uv} is associated with the edge $[u, v]$. In road-networks, this is the distance or the travel time between intersections u and v. N refers to the space of points located on the edges/vertices of graph G. For a point $p \in N$ located on the edge $[u, v]$, $w_{up} = \frac{|up|}{|uv|} w_{uv}$ where $|uv|$ is the Euclidean distance between u and v. Figure 1a shows the graph model of a road network including the vertex set $V = \{a,...,p\}$. Each edge of the graph is labeled by its weight. Figure 1b shows points $s_1,...,s_4$, $r_1,...,r_3$, and $t_1,...,t_3$ on the edges of the same graph. As shown in the figure, point $r_1 \in N$ corresponds to the weights $w_{r_1 a} = 1$ and $w_{r_1 b} = 3$.

The main challenge with query processing in road-networks is the expensive cost of computing the



Road Networks. Figure 1. (a) Graph model of a road network, (b) 10 points of interest on the edges of the graph.

network distance between two points. Hence, *network distance* for road-networks is formally defined below.

Definition 1: Given a graph G, a *path* P from $p_1 \in N$ to $p_2 \in N$ is an ordered set $P = \{p_1, v_1,...,v_n, p_2\}$ consisting of a sequence of connected edges from p_1 to p_2. Here, p_1 and p_2 are located on the edges $[u, v_1]$ and $[v_n, w]$, respectively. Also, v_i is connected to v_{i+1} by the edge $[v_i, v_{i+1}]$ for $1 \leq i < n$. As shown in Fig. 1b, $P = \{t_1, a, b, s_3\}$ is a path from t_1 to s_3.

Definition 2: Given a path $P = \{p_1, v_1,...,v_n, p_2\}$, *Path Cost* of P, $pcost(P)$, is defined as the sum of the costs of all edges in P. Formally, for the path P,

$$pcost(P) = w_{p_1 v_1} + \sum_{i=1}^{n-1} w_{v_i v_{i+1}} + w_{v_n p_2}$$

In Fig. 1b, the cost of path $P = \{t_1, a, b, s_3\}$ is calculated as $pcost(P) = 3 + 4 + 1 = 8$. For the points $p_1, p_2 \in N$, $P_{p_1 p_2}$ is used to denote the *shortest path* from p_1 to p_2 in G; the path $P = \{p_1,...,p_2\}$ with minimum cost $pcost(P)$.

Definition 3: Given the two points p_1 and p_2 in N, the *network distance* between p_1 and p_2, $D_n(p_1, p_2)$, is the cost of the shortest path between p_1 and p_2 (i.e., $D_n(p_1, p_2) = pcost(P_{p_1 p_2})$). For instance, $D_n(t_1, s_3) = 8$. The network distance $D_n(.,.)$ is non-negative and obeys identity, symmetry and the triangular inequality. Hence, together with N, it forms a metric space.

As discussed in the background section several techniques have been proposed to expedite the computation of this network-distance for efficient processing of distance-based queries. Here, one hybrid approach is reviewed which combines indexing the space (using voronoi diagrams) with a distance pre-computation approach and has shown to be superior in performance to its competitors [8].

A Voronoi-Based Solution for Road-Networks

A comprehensive solution for spatial queries in road-networks must fulfill the following real-world requirements: (i) be able to incorporate the network connectivity to provide exact distances between objects, (ii) efficiently answer the queries in real-time in order to support distance-based queries (e.g., *k*NN) for moving objects, (iii) be scalable in order to be applicable to usually very large networks, (iv) be independent of the density and distribution of the points of interest, (v) be adaptive to efficiently cope with database updates where nodes, links, and points of interest are added/deleted, and (vi) be extendible to consider query constraints such as direction or range.

In [8], the authors proposed a novel approach that fulfills the above requirements by reducing the problem of distance computation in a very large network, in to the problem of distance computation in a number of much smaller networks plus some additional table lookups.

The main idea behind this approach, termed Voronoi-based Network Nearest Neighbor (VN3), is to first partition a large network in to smaller/more manageable regions. This is achieved by generating a first-order *network* Voronoi diagram over the points of interest. Each cell of this Voronoi diagram is centered by one point of interest (e.g., a restaurant) and contains the nodes that are closest to that object in *network* distance (and not the Euclidean distance). Next, the intra and inter distances for each cell are pre-computed. That is, for each cell, the distances between all the edges (or border points) of the cell to its center are pre-computed. In addition, the distances across the border points of the *adjacent* cells are also pre-computed. This will reduce the pre-computation time and space by localizing the computation to cells and handful of neighbor-cell node-pairs.

Now, to find the k nearest-neighbors of a query object q, first the first nearest neighbor is found by simply locating the Voronoi cell that contains q. This can be easily achieved by utilizing a spatial index (e.g., R-tree) that is generated for the Voronoi cells. In [8], it is shown that the next nearest neighbors of q are within the adjacent cells of the previously explored ones, which can be efficiently retrieved from a lookup table. Next, the intra-cell pre-computed distances are utilized to find the distance from q to the borders of the Voronoi cell of each candidate, and finally the inter-cell pre-computed distances are used to compute the actual network distance from q to each candidate. The local pre-computation nature of VN3 also results in low complexity of updates when the network is modified.

Key Applications

The applications of distance-based queries on road-networks are numerous. In emergency response, the first responders may want to find k closest hospitals to a crisis area (kNN query). In urban planning, one may

need to find the set of parks that are closest to a set of houses (known as *spatial skyline queries*). In location-based services, a group of mobile users want to find a meeting location where traveling towards which minimizes their total travel distance (Aggregate-NN query). In Location-based Services (LBS), a driver wants to find all the gas stations within its 4-mile distance (spatial range queries). In OnLine map services such as Yahoo! Maps, Google Earth or Microsoft Virtual Earth, one may want to minimize distance when planning a day trip to a shopping center, a restaurant and a movie theater (OSR query). In all these applications and queries, the accurate and fast computation of network distance given the underlying road network is critical.

Future Directions

Even though, as reviewed in this entry, several techniques for fast computation of network distances have been proposed, there are still no real-world deployment of these or any other techniques to enable accurate and fast network distance computation for spatial queries. This may be attributed to the fact that many users can tolerate or are used to the inaccuracy in distance computations. However, as the collected geospatial data becomes more accurate and the users become more sophisticated, the competition between different geospatial services would lead into the adaptation of a simple but effective technique to provide accurate network distance for query processing. Therefore, a technique that can easily be integrated into the current information infrastructure used by geospatial applications is of substantial importance.

On the research front, new queries and applications for multidimensional spaces are often proposed in academic conferences and journals. Most of these queries are applicable to geospatial applications and road-networks. Therefore, a rather effortless way of finding a new research topic is to take any of these new queries and extend it to work in the road-network space.

Finally, adding other attributes that would affect the distance or time-to-travel in road-networks renders most if not all of the current solutions insufficient. For example, one can consider efficient on-the-fly computation of network distance in the presence of traffic flow, road direction, road closure, road elevation, or navigational data such as costs of left-turns, right turns, U-turns, and stops.

Cross-references

- ▶ Rtree
- ▶ Spatial Data Analysis
- ▶ Spatial Data Mining
- ▶ Spatial Data Types
- ▶ Spatial Indexing Techniques
- ▶ Spatial Network Databases
- ▶ Spatial Operations and Map Operations
- ▶ Spatio-Temporal Trajectories
- ▶ Voronoi Diagrams

Recommended Reading

1. Almeida V.T.D. and Güting R.H. Indexing the Trajectories of Moving Objects in Networks, GeoInformatica, 9(1):33–60, 2005.
2. Cho H.-J. and Chung C.-W. An efficient and scalable approach to cnn queries in a road network. In Proc. 31st Int. Conf. on Very Large Data Bases, 2005, pp. 865–876.
3. Güting H., de Almeida T., and Ding Z. Modeling and querying moving objects in networks. VLDB J., 15(2):165–190, 2006.
4. Hu H., Lee D.L., and Lee V.C.S. Distance indexing on road networks. In Proc. 32nd Int. Conf. on Very Large Data Bases, 2006, pp. 894–905.
5. Hu H., Lee D.L., and Xu J. Fast nearest neighbor search on road networks. In Advances in Database Technology, Proc. 10th Int. Conf. on Exdending Database Technology, 2006, pp. 186–203.
6. Huang X., Jensen C.S., and Saltenis S. The islands approach to nearest neighbor querying in spatial networks. In Proc. 9th Int. Symp. Advances in Spatial and Temporal Databases, 2005, pp. 73–90.
7. Jensen C.S., Kolářvr J., Pedersen T.B., and Timko. I. Nearest neighbor queries in road networks. In Proc. 11th ACM Int. Symp. on Advances in Geographic Inf. Syst., 2003, pp. 1–8.
8. Kolahdouzan M.R. and Shahabi C. Voronoi-based k nearest neighbor search for spatial network databases. In Proc. 30th Int. Conf. on Very Large Data Bases, 2004, pp. 840–851.
9. Ku W.-S., Zimmermann R., Wang H., and Wan C.-N. Adaptive nearest neighbor queries in travel time networks. In Proc. 13th ACM Int. Symp. on Geographic Inf. Syst., 2005, pp. 210–219.
10. Papadias D., Zhang J., Mamoulis N., and Tao Y. Query processing in spatial network databases. In Proc. 29th Int. Conf. on Very Large Data Bases, 2003, pp. 790–801.
11. Pfoser D. and Jensen C.S. Indexing of network constrained moving objects. In Proc. 11th ACM Int. Symp. on Advances in Geographic Inf. Syst., 2003, pp. 25–32.
12. Sankaranarayanan J., Alborzi H., and Samet H. Efficient query processing on spatial networks. In Proc. 13th ACM Int. Symp. on Geographic Inf. Syst., 2005. pp. 200–209.
13. Shahabi C., Kolahdouzan M.R., and Sharifzadeh M. A road network embedding technique for k-nearest neighbor search in moving object databases. In Proc. 10th ACM Int. Symp. on Advances in Geographic Inf. Syst., 2002, pp. 94–100.

14. Sharifzadeh M. and Shahabi C. Processing optimal sequenced route queries using voronoi diagrams. GeoInformatica, 12 (4):411–433, 2008.
15. Yiu M.L., Mamoulis N., and Papadias D. Aggregate nearest neighbor queries in road networks. IEEE Trans. Knowl. Data Eng., 17(6):820–823, 2005.

Road Vector Data

► Road Networks

Robot

► Web Crawler Architecture

ROC

► Receiver Operating Characteristic

Rocchio's Formula

Ben He
University of Glasgow, Glasgow, UK

Definition

Rocchio's formula is used to determine the query term weights of the terms in the new query when Rocchio's relevance feedback algorithm is applied.

Key Points

In 1971, Rocchio proposed a classical query expansion algorithm based on the Vector Space model [1]. The basic algorithm assumes that the user identifies a set R of relevant documents and a set N of non relevant documents and the improved query is the result of a linear combination of the mean frequencies tf of the terms in the original query and in these two sets (*the centroids of R and N*), that is the weight of each term in the new query is:

$$qtf_m = \alpha.qtf + \beta.\sum_{d\in R} tf - \gamma.\sum_{d\in N} tf$$

Feedback variations assume that positive feedback exhibits a much clear impact on the reformulation of the query. Also positive feedback can be applied to expand query without the explicit feedback from the user (*blind relevance feedback*). Blind relevance feedback can be obtained in four steps:

1. All documents are ranked for the given query using a particular Information Retrieval model, for example the TF-IDF term weighting of the vector space model. This step is called *first-pass retrieval*. The user identifies a set R of relevant documents and a set N of non relevant documents.
2. A weight qtf_{exq} is assigned to each term appearing in the set of the k highest ranked documents. In general, qtf_{exq} is the mean of the weights provided by the Information Retrieval model, for example the TF-IDF weights, computed over the set of the k highest ranked documents.
3. The vector of query terms weight is finally modified by taking a linear combination of the initial query term weights qtf used for the first-pass retrieval and the new weight qtf_{exq}, that is:

$$qtw_m = qtf + \beta \cdot qtf_{exq} \quad (1)$$

4. To remove noisy terms from the expanded query, automatic query expansion techniques usually selects only the highest informative terms from the set of top-ranked documents. The informativeness of a term is determined by the terms with highest weights assigned in step 2.

Cross-references

► Query Expansion Models
► Relevance Feedback

Recommended Reading

1. Rocchio J. Relevance Feedback in Information Retrieval. Prentice-Hall, Englewood Cliffs, NJ, USA, 1971, pp. 313–323.

Role Based Access Control

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

Synonyms

RBAC; Role based security

Definition

Access control is a security service responsible for defining which subjects can perform what type of operations on which objects. A subject is typically an active entity such as a user or a process, and an object is an entity, such as a file, database table or a field, on which the subject can perform some authorized operations. A permission indicates the mode of operation on a particular object.

Role based access control (RBAC) involves controlling access to computer resources and information by (i) defining *users*, *roles*, and *permissions*, and (ii) assigning users and permissions to roles. A user can create a *session* in which he/she can activate a subset of the roles he/she has been assigned to and use the permissions associated with the activated roles. RBAC approach is based on the understanding that a user's access needs are defined by the roles that he/she plays within his/her organization. In general, a role is considered as a group of permissions. RBAC approach also uses *role-role relation*, known as *role-hierarchy*, to provide permission inheritance semantics, and *constraints* on the assignment relations and the activation of roles to capture various access control requirements.

Historical Background

The origin of the concept of roles can be traced back to organizational theory much earlier than the advent of computerized information systems. However, it was primarily in the early 1990s that the security researchers and practitioners became interested in adopting the notion of a role to address the access control issues for information systems. In particular, in 1992, Ferraiolo and Kuhn showed that the existing mandatory and discretionary access control (MAC and DAC) approaches were inadequate in addressing the complex and diverse access control needs of various organizations and proposed the use of a Role based approach. While the MAC approach uses the predefined set of system rules to control accesses to resources, thus not giving users discretionary power to grant the rights they have on objects (e.g., files, databases) to other users/subjects, the DAC approach allowed the users to grant the permissions that they have on objects to others freely. Later, the initial work by Ferrailolo and Kuhn was followed by Nyanchama and Osborn's role graph model in 1995 [9], and later by the seminal paper by Sandhu, Coyne, Feinstein and Youman in 1996 [13], where they defined a family of RBAC models each with different sets of capabilities, known as the RBAC96 model. This model later evolved into the NIST RBAC model. The NIST model was later modified into the ANSI/INCITS (ANSI/ICITS stands for American National Standards. Institute and International Committee for Information Technology Standards) RBAC standard in 2004 [2]. The significant interest in this area can be seen by the establishment of the ACM Workshop on RBAC in 1996 which later evolved into the current ACM Symposium in Access Control Method and Technologies (SACMAT). Several extensions of the RBAC model have been proposed and several newer RBAC issues have been identified over the last one decade. A key contribution is also related to the demonstration by Osborn et al. that the RBAC96 model can also be configured to represent the MAC and DAC policies [10], establishing its usefulness as a uniform model for addressing very diverse set of access control needs.

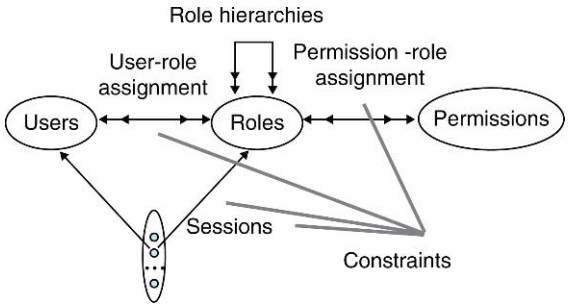
Foundations

Role is a prevalent organizational concept and it identifies the various job functions and responsibilities within an organization. As organizational information systems has become a crucial component for carrying out organizational job functions, it has motivated the use of roles that users within the organization play to define what accesses should be authorized to them so that they can carry out their job functions and responsibilities efficiently. Based on the premise that the role represents the set of permissions that are needed for carrying out the job functions, various RBAC approaches have been proposed. RBAC96 model is the most widely recognized initial model which has evolved into the ANSI/ICITS RBAC standard. The RBAC96 family of models is briefly described next followed by a discussion on other extensions made to it and its standardized version.

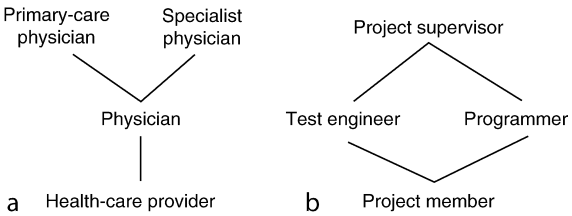
Figure 1 depicts the RBAC96 family of models which include $RBAC_0$, $RBAC_1$, $RBAC_2$, and $RBAC_3$. $RBAC_0$ is the base model containing only basic elements; $RBAC_1$ augments $RBAC_0$ with role hierarchy; $RBAC_2$ augments $RBAC_0$ with constraints; and $RBAC_3$ combines $RBAC_1$ and $RBAC_2$, and provides the most complete set of features.

$RBAC_0$ Base Model

In the base model $RBAC_0$, the key elements are the sets users (U), roles(R), permissions (P) and sessions (S).



Role Based Access Control. Figure 1. RBAC96 model.



Role Based Access Control. Figure 2. Sample Role Hierarchies.

Various relations are defined among them, as depicted in Fig. 1. A *user* in this model is a human being but can also be generalized to include intelligent autonomous agents such as robots, immobile computers, or even networks of computers. A *role* typically represents a job function within the organization with some authority and responsibility. The objects are data objects as well as other computer resources within a computer system. RBAC96 model only supports "positive permissions" that grants accesses to objects, and does not support "negative permissions" that deny accesses. Constraints, as defined in $RBAC_2$, are used as a mechanism to achieve the denial of accesses.

The *user assignment* (*UA*) and *permission assignment* (*PA*) relations shown in Fig. 1 are many-to-many relations. A user can be a member of many roles, and a role can have many users. Similarly, a role can have many permissions and the same permission can be assigned to many roles. The placement of a role as an intermediary to enable a user to exercise a permission provides much greater control over access configuration and review than does directly relating users to permissions.

A *session* is essentially a mapping of one user to possibly many roles, i.e., a user can establish a session and activate within it some subset of roles that he/she has been assigned to. The set of permissions that can be used by a user is the union of the permissions assigned to the roles that the user activates in one session. The association between a user and the activated set of roles within a session remains constant for the life of the session. A user may have multiple sessions open at the same time and each session may have a different combination of assigned roles activated in it. This feature of $RBAC_0$ supports the principle of least privilege that involves use of the minimal set of permissions needed for a particular task. A user who is a member of several roles can invoke any subset of these within a session to complete a task. The concept of a session is equivalent to that of the traditional notion of a *subject* in the access control literature. A subject (or session) is a unit of access control, and a user may have multiple subjects (or sessions) with different permissions [13].

$RBAC_1$: RBAC with Role Hierarchy

$RBAC_1$ introduces role hierarchies (*RH*) on top of $RBAC_0$ to indicate which roles can inherit which permissions from which other roles. Role hierarchies are a natural way of structuring roles to reflect an organization's lines of authority and responsibility. Figure 2 depicts some example of role hierarchies. By convention senior roles are shown toward the top of these diagrams, and junior roles toward the bottom. For example, in Fig. 2(a), the junior-most role is *health-care provider*. The *physician* role is senior to *health-care provider* and thereby inherits all permissions from *health-care provider*. In addition to the permissions inherited from the *health-care provider* role, the *physician* role can have other permissions assigned directly to it. Inheritance of permissions is transitive. For example, in Fig. 2(a), the *primary-care physician* role inherits permissions from the *physician* and *health-care provider* roles. *Primary-care physician* and *specialist physician* both inherit permissions from the *physician* role. Figure 2(b) illustrates multiple inheritances of permissions, where the *project supervisor* role inherits from both *test engineer* and *programmer* roles. Mathematically, these hierarchies are partial orders. A partial order is a reflexive, transitive and anti-symmetric relation.

$RBAC_2$: RBAC with Constraints

$RBAC_2$ introduces the concept of constraints. Constraints are an important aspect of RBAC and are sometimes argued to be a principal motivation for using RBAC. Constraints are a powerful mechanism for capturing higher-level organizational security

policies. Constraints can be applied to the *UA* and *PA* relations, as well as to the association between a user and its activated set of roles within a session.

The most frequently mentioned constraint in the context of RBAC is the *mutually exclusive roles* (MER) constraint that is used to enforce Separation of Duty (SoD) requirements. For instance, a user can be restricted to assume at the most one role in a mutually exclusive role set. This supports separation of duty requirements where a person should not be able to assume two different roles to carry out critical steps of a task. Consider two mutually exclusive roles, *accounts-manager* and *purchasing-manager*. Mutual exclusion in terms of *UA* specifies that one individual cannot be a member of both these roles. Mutual exclusion in terms of *PA* specifies that the same permission cannot be assigned to both these roles. For example, the permission to issue checks should not be assigned to both these roles. Normally such a permission would be assigned to the *accounts-manager* role only. The mutual exclusion constraint on *PA* helps prevent the permission from being unintentionally or intentionally assigned to the *purchasing-manager* role.

Another example of a user assignment constraint is that a role can have a maximum number of members. For instance, the number of roles to which and individual user can belong to could also be limited. These are called *cardinality constraints*. Similarly, the number of roles to which a permission can be assigned can have cardinality constraints to control the distribution of powerful permissions.

$RBAC_3$: The Consolidated Model

$RBAC_3$ combines $RBAC_1$ and $RBAC_2$ to provide both role hierarchies and constraints. There are several issues that arise by bringing these two concepts together.

Constraints can be applied to the role hierarchy itself. For example, constraints on *RH* can limit the number of senior (or junior) roles that a given role may have. Two or more roles can also be constrained to have no common senior (or junior) role. These kinds of constraints are useful in situations where the authority to change the role hierarchy has been decentralized, but the chief security administrator desires to restrict the cases in which such changes can be made.

Benefits of the RBAC Approach

The RBAC approach has been recognized for several beneficial features. First, it supports the principle of least privilege which has been considered very important for better security of systems. When RBAC is used, a user's access requirements can be easily changed when his role is changed within his organization. All that should be done is removing him as a member of his current role and assigning him to the new role. Also, the total number of assignment relationships that needs to be maintained in RBAC is $n_r(n_u + n_p)$ where n_r, n_u, and n_p represent the number of roles, users and permissions, respectively, in a system. In general subject-object based authorization system will need to maintain $n_u.n_p$ subject-to-permission associations. Role hierarchy makes security administration significantly easy as it eliminates the need for explicitly assigning the permissions to multiple roles. Constraints can be used to capture various types of policies including very important SoD requirements. It has been shown that by configuring constraints and relationships in RBAC, one can configure an RBAC system to express the traditional DAC and MAC policies [10]. Because of its capability to capture very diverse sets of requirements, it has been considered as a very promising approach to address emerging multidomain security problems where various domains with different access control policies need to securely interact with each other.

RBAC Standards

The NIST RBAC model was the first attempt towards establishing an RBAC standard. Compared to the RBAC96 models, the most distinct feature in NIST RBAC is the clear specification of incremented 4-level RBAC models. The first level, called the core RBAC is similar to the $RBAC_0$ model with the explicit specification of user-role view where the roles assigned to a specific user and the users assigned to a specific role can be determined. The second level, called the hierarchical RBAC model is based on the $RBAC_1$ model and includes the separation of general hierarchy (the same as in $RBAC_1$) and the restricted hierarchy, where the structure of the hierarchy is restricted to a certain type, such as, a tree or an inverted tree structure. The third level is the same as the constraint RBAC. The fourth level further adds the permission-role view where the roles assigned to a specific permission and the permissions assigned to a specific role can be determined. Later, the NIST model was further refined to establish the ANSI/INCITS RBAC standard.

Administration Models for RBAC

In large organizational systems the number of roles can be in the hundreds or thousands. Managing these roles and their relationships is a daunting task that is often highly centralized. A key issue is how the benefits of the RBAC model can be used to construct an administration model for managing the RBAC policies. Several researchers have addressed this issue, including the Administrative RBAC (ARBAC) family of models by Sandhu et al. [12] and Scoped ARBAC (SARBAC) family of models by Crampton et al. [6].

RBAC Extensions

The RBAC96 model and its standardized versions also have been extended in several ways to address emerging applications. One notable among the emerging requirements is the need to capture context based access control requirements. Several extensions of RBAC models have been made to address such a need. Notable among these are extensions of RBAC to capture temporal and/or location context. Temporal RBAC (TRBAC) model [3] and its generalized version Generalized TRBAC (GTRBAC) [7] are main work related to temporal extensions of the RBAC model. While these capture the time context, several works have tried to address the need of capturing location context within an RBAC framework. Covington et al. propose a Generalized RBAC (GRBAC) model, where they introduce the concept of environment roles, which are roles that can be activated based on the value of conditions in the environment where the request has been made. Bertino et al. [4] has recently proposed the GEO-RBAC model that integrates RBAC with a spatial model based on the OpenGIS system. A significant aspect of the GEO-RBAC model includes (i) its specification of role schema that are location-based, and (ii) the separation of role schema and role instances to provide different authorizations for different logical and physical locations. Location and time-based RBAC (LoT-RBAC) model was proposed by Chandran et al. [5] to address the access control requirements of highly mobile, dynamic environments to provide both location and time based control. Significant work has also been done to try to develop specification languages for RBAC models. This includes the inclusion of RBAC profile in OASIS's XACML. Joshi et al. propose X-RBAC that generalizes X-GTRBAC, which is the XML-based language for specifying the GTRBAC policies. Other extensions to the RBAC work include developing better constraint frameworks and analyzing their complexity issues.

Support for SoD constraints is a major issue for the RBAC model. Various SoD constraints proposed in the literature include: Static SoD, Dynamic SoD, History Based SoD, Object Based SoD, Operational SoD, Order Dependent/Independent SoD. The ANSI RBAC standard, however, only supports the basic SoD constraints. Several researchers have developed RBAC extensions to support these advanced SoD constraints. Ahn et al. have proposed the RCL2000 language for specifying several role-based SoD constraints [1]. Joshi et al. have proposed time-based SoD constraints in [8].

RBAC approach has also been shown to be very beneficial for multi-domain security which refers to the need to facilitate secure interactions among multiple security domains with very diverse set of access control requirements. Several researchers have recently focused on using RBAC to address the multi-domain security challenge [11,14]. One approach uses mapping roles from multiple domains to ensure secure cross--domain accesses. Such inter-domain role mapping problem has been recently formalized and solutions for both tightly coupled environments as well as lightly coupled environments are being sought [11,14]. Some researchers have employed RBAC delegation models to address such multi-domain sharing issues [15]. Several RBAC delegation models have been proposed that use roles as the central entity in the process of delegating rights from one user to another.

Key Applications

RBAC has been used in operating systems, databases and applications. In particular, RBAC is very promising for large scale application environments and enterprises. Emerging systems and applications have more context and content based access control requirements as can be seen in mobile and peer to peer applications and several extensions of RBAC has been motivated by such requirements.

Future Directions

Future applications will require efficient access control techniques to protect large number of objects and manage huge number of privileges that may need to be given to unknown users. Furthermore, interactions among multiple domains with different access control policy requirements are crucial in emerging applications. RBAC has been found to be promising for

these emerging access control requirements. Several researchers have advocated use of RBAC to address large scale enterprise security as well as multidomain security. Work related to multidomain security using RBAC approach is in its initial stages of development. Policy verification and evolution management issues are crucial for access control in large and dynamic systems but little work currently have addressed these.

URL to Code

NIST provides an implementation of some RBAC versions http://csrc.nist.gov/rbac/.
ANSI/INCITS website: http://csrc.nist.gov/groups/SNS/rbac/standards.html.

Cross-references

- ▶ Access Control Policy Languages
- ▶ Discretionary Access Control
- ▶ Mandatory Access Control
- ▶ Security Policies
- ▶ Temporal Access Control

Recommended Reading

1. Ahn G. and Sandhu R. Role-based authorization constraints specification. ACM Trans. Inf. Syst. Secur., 3(4):207–226, 2000.
2. American national standard for information technology (ANSI). Role based access control. ANSI INCITS 359–2004, February 2004.
3. Bertino E., Bonatti P.A., and Ferrari E. TRBAC: a temporal role-based access control model. ACM Trans. Inf. Syst. Secur., 4(3):191–233, 2001.
4. Bertino E., Catania B., Damiani M.L., and Perlasca P. GEO-RBAC: A spatially aware RBAC. In Proc. 10th ACM Symp. on Access Control Models and Technologies, 2005, pp. 29–37.
5. Chandran S.M. and Joshi J.B.D. LoT RBAC: a location and time-based RBAC model. In Proc. 6th Int. Conf. on Web Information Systems Eng., 2005, pp. 361–375.
6. Crampton J. and Loizou G. Administrative scope: a foundation for role-based administrative models. ACM Trans. Inf. Syst. Secur., 6(2):201–231, 2003.
7. Joshi J.B.D., Bertino E., Latif U., and Ghafoor A. A generalized temporal role-based access control model. IEEE Trans. Knowl. Data Eng., 17(1):4–23, 2005.
8. Joshi J.B.D., Shafiq B., Ghafoor A., and Bertino E. Dependencies and separation of duty constraints in GTRBAC. In Proc. 8th ACM Symp. on Access Control Models and Technologies, 2003, pp. 51–64.
9. Nyanchama M. and Osborn S.L. The role graph model. In Proc. 1st ACM Workshop on Role-Based Access Control, 1995.
10. Osborn S., Sandhu R., and Munawer Q. Configuring role-based access control to enforce mandatory and discretionary access control policies. ACM Trans. Inf. Syst. Secur., 3:85–106, 2000.
11. Piromruen S. and Joshi J.B.D. An RBAC framework for time constrained secure interoperation in multi-domain environment. In Proc. IEEE Workshop on Object-oriented Real-time Dependable Systems, 2005, pp. 36–45.
12. Sandhu R., Bhamidipati V., and Munawer Q. The ARBAC97 model for role-based administration of roles. ACM Trans. Inf. Syst. Secur., 2(1):105–135, 1999.
13. Sandhu R.S., Coyne E.J., Feinstein H.L., and Youman C.E. Role-based access control models. IEEE Comput., 29(2):38–47, 1996.
14. Shafiq B., Joshi J.B.D., Bertino E., and Ghafoor A. Secure interoperation in a multi-domain environment employing RBAC policies. IEEE Trans. Knowl. Data Eng., 17(11):1557–1577, 2005.
15. Zhang L., Ahn G., and Chu B. A role-based delegation framework for healthcare information systems. In Proc. 7th ACM Symp. on Access Control Models and Technologies, 2002, pp. 125–134.

Role Based Security

- ▶ Role Based Access Control

Rollback

- ▶ Crash Recovery
- ▶ Logging and Recovery

Rollback Operator

- ▶ Timeslice Operator

Rotation

- ▶ Dynamic Graphics

Rotation Estimation

- ▶ Cross-Validation

Rough Computing

- ▶ Decision Rule Mining in Rough Set Theory

Rough Set Theory (RST)

► Decision Rule Mining in Rough Set Theory

Rough Set Theory, Granular Computing on Partition

► Deductive Data Mining Using Granular Computing

Rounding

► Microdata Rounding

Row-Level Locking

► B-Tree Locking

Row-Versioning

► Snapshot Isolation

R-Precision

Nick Craswell
Microsoft Research Cambridge, Cambridge, UK

Definition

For a given query topic Q, R-precision is the precision at R, where R is the number of relevant documents for Q. In other words, if there are r relevant documents among the top-R retrieved documents, then R-precision is $\frac{r}{R}$.

Key Points

R-precision is defined as the proportion of the top-R retrieved documents that are relevant, where R is the number of relevant documents for the current query. This requires full knowledge of a query's relevant set, and will be a shallow evaluation for a query with few relevant documents and a deep evaluation for a query with many relevant documents. Rank cutoff R is the point at which precision and recall are equal, since at that point both are $\frac{r}{R}$.

Cross-references

► Average R-Precision
► Precision
► Precision at n
► Precision-Oriented Effectiveness Measures

RSJ Model

► Probabilistic Retrieval Models and Binary Independence Retrieval (BIR) Model

Rtree

► Tree-based Indexing

R-Tree (and Family)

Apostolos N. Papadopoulos[1], Antonio Corral[2], Alexandros Nanopoulos[1], Yannis Theodoridis[3]
[1]Aristotle University, Thessaloniki, Greece
[2]University of Almeria, Almeria, Spain
[3]University of Piraeus, Piraeus, Greece

Definition

The R-tree is an indexing scheme that has been originally proposed towards organizing spatial objects such as points, rectangles and polygons. It is a hierarchical data structure suitable to index objects in secondary storage (disk) as well as in main memory. The R-tree has been extensively used by researchers to offer efficient processing of queries in multi-dimensional data sets. Queries such as *range*, *nearest-neighbor* and *spatial joins* are supported efficiently leading to considerable decrease in computational and I/O time in comparison to previous approaches. The R-tree is capable of handling diverse types of objects, by using approximations. This means that an object is approximated by its minimum bounding rectangle (MBR) towards providing an efficient filtering step. Objects that

survive the filtering step are inspected further for relevance in the refinement step. The advantages of the structure, its simplicity as well as its resemblance to the B^+-tree "persuaded" the database industry to implement it in commercially available systems in addition to research prototypes.

Historical Background

The R-tree index was proposed by Guttman [6] in 1984 in order to solve an organization problem regarding rectangular objects in VLSI design. Later on, the structure was revised to become even more efficient and to adapt to the particular problem. The set of variants comprise the so called *R-tree family* of access methods.

The first variant, the R^+-tree, was proposed in 1987 [15]. The main difference between the two schemes is that while in the R-tree the MBR of an object is placed to one leaf node only, in the R^+-tree the MBR may break to several smaller MBRs and stored in different leaf nodes. The motivation behind this new index is that large MBRs may cause performance degradation. However, storage utilization decreases because of the MBR decomposition.

The next most important variation, the R^*-tree [1], retains the properties of the original R-tree and provides better strategies for inserting MBRs. New heuristics are offered towards improving the shape of the tree during insertions and node splits. Performance evaluation results have shown that the R^*-tree offers significantly better performance in *range query* processing in comparison to R-trees and R^+-trees. The trade-off is that more time is spent during insertions and deletions of objects, since the heuristics used require more computational overhead.

Another important variation of the R-tree is the Hilbert R-tree [8]. It is a hybrid structure based on the R-tree and the B^+-tree. Actually, it is a B^+-tree with geometrical objects being characterized by the Hilbert value of their centroid. The structure is based on the Hilbert space-filling curve. According to the authors' experimentation in [8], Hilbert R-trees were proven to be the best dynamic version of R-trees as of the time of publication. The term *dynamic* is used to denote that insertions and deletions are allowed in the underlying data set.

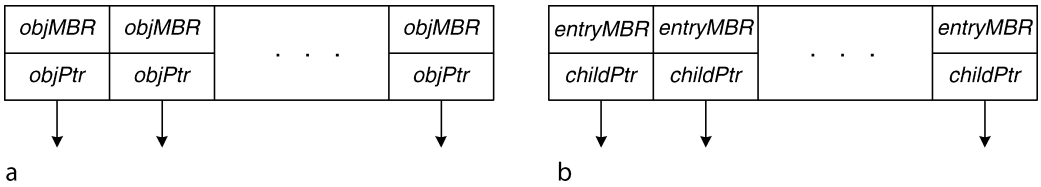
The aforementioned contributions focused on the use of heuristics during tree construction with the aim to provide a well-formed structure towards better query processing. However, by inserting objects one-by-one two problems may arise: (i) the shape of the structure may not be compact and (ii) the storage utilization will never reach 100% due to the rules applied in node splits. In cases where objects are known in advance, a bottom-up building process (also called bulk-loading) may be applied in contrast to the usual top-down tree construction. The first contribution in this area is due to Roussopoulos and Leifker in [13] who proposed the use of packed R-trees. Along the same lines, Kamel and Faloutsos [7] proposed a packed version of the R-tree where packing is performed by the use of the Hilbert space filling curve. Other packed variants have been proposed in [3,4,9].

The R-tree inspired subsequent work on handling high-dimensional data. It has been observed by many researchers that the performance of R-trees degrades rapidly when the number of dimensions increases above a threshold (10 or 15). To tackle this dimensionality curse problem a number of R-tree variations appeared that show better scaling capabilities for high-dimensional spaces. Some of these contributions are the TV-tree [10], the X-tree [2] and the A-tree [14].

Due to space limitations, the description of more variations (actually these are more than 70) is not feasible. A more detailed presentation of the R-tree family can be found in [11].

Foundations

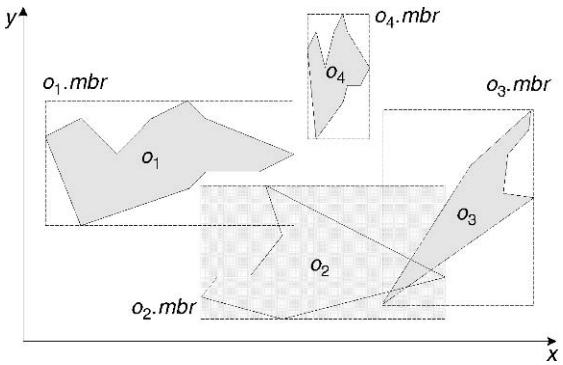
Let $\mathcal{O}$ be a set of objects in the two-dimensional space. Each object $o_i \in \mathcal{O}$ is represented by its MBR, which is the minimum rectangle that completely encloses the object. The MBR of object o_i is denoted by $o_i.mbr$. Since MBRs are forced to be orthogonal with respect to the axes, each MBR is completely defined by its lower-left and upper-right corners. The R-tree organizes data in a hierarchical manner. It is a height-balanced tree where object MBRs are hosted in leaf nodes, whereas internal nodes contain auxiliary entries to guide the search process. More formally, each leaf node contains pairs of the form (*objMBR*, *objPtr*) where *objMBR* is the MBR of an object, and *objPtr* is the pointer to the object's detailed information (geometry and/or other attributes). Each internal node contains entries of the form (*entryMBR*,*childPtr*) where *entryMBR* is the MBR enclosing all descendants in the corresponding subtree and *childPtr* is the pointer to the subtree. The format of leaf and internal nodes is illustrated in Fig. 1.



R-Tree (and Family). Figure 1. Format of leaf and internal nodes. (a) Format of a leaf node. (b) Format of an internal node.

Each R-tree node corresponds to one disk page which has a limited capacity. The number of entries in each internal node determines the tree *fanout*, which is the maximum number of subtrees that can emanate from an internal node. Let M_{int} and M_{leaf} denote the maximum number of entries that can be hosted in an internal node and a leaf node, respectively. To guarantee acceptable storage utilization, a minimum number of entries per node must also be defined. Therefore, the tree construction procedure forces that each internal node (leaf) must contain no less than m_{int} (m_{leaf}) entries. In the case where the R-tree stores MBR of objects, then evidently $M_{int} = M_{leaf}$ and $m_{int} = m_{leaf}$. However, there are cases where the above equalities do not hold. For example, when the R-tree stores point objects, then the number of objects in a leaf node increases, since a point requires less data for representation than a rectangle. For ease of illustration, for the rest of the article it is assumed that the R-tree stores rectangles, which means that the minimum and the maximum number of entries is the same for all tree nodes (the symbols m and M are used, respectively). Therefore, each tree node contains at least $m \geq M/2$ entries and at most M entries. A violation of this rule is only allowed for the root of the tree, which may contain less than m entries.

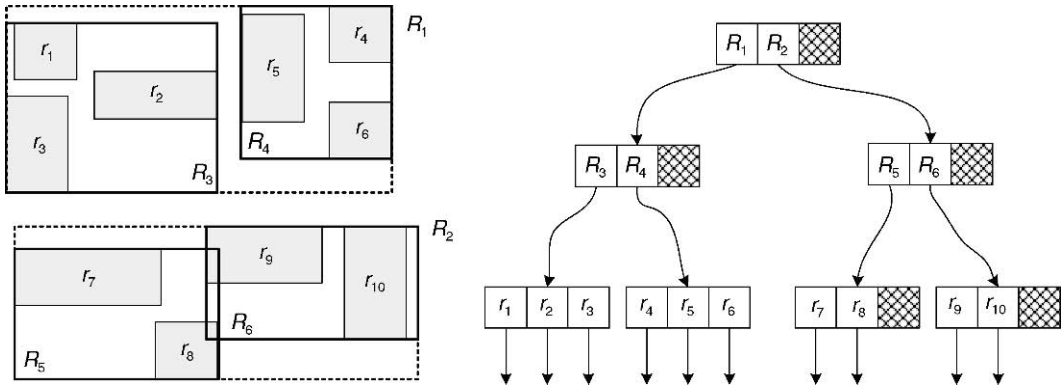
Since MBR is an approximation of the original object, intersection tests must be performed in two steps to guarantee correctness of results. Figure 2 depicts three polygonal objects with their corresponding MBRs. Searching for objects intersected by o_2, it is evident that both o_1 and o_3 may be contained in the final answer, because $o_2.mbr$ intersects both $o_1.mbr$ and $o_3.mbr$. This information is easily obtained by performing efficient intersection tests for rectangular objects. However, to produce the final result the answer requires further refinement by considering the detailed geometric characteristics of objects o_1, o_2 and o_3. This refinement step involves more complex geometric computations and it is applied only for candidate objects. Evidently, if an MBR does not intersect $o_2.mbr$ (e.g., $o_4.mbr$) there is no need to investigate it further.



R-Tree (and Family). Figure 2. Intersection of objects and MBRs.

Figure 3 shows a set of objects in the 2-d space (left) and the corresponding R-tree index (right). For simplicity only the MBRs of objects are shown using the symbols r_1 through r_{12}. The R-tree is composed of seven nodes. There are four leaf nodes containing the object MBRs, and three internal nodes containing MBRs that enclose all the descendants in the corresponding subtree. For example, the MBR R_1 which is hosted at the root, encloses MBRs R_3 and R_4. Moreover, R_3 encloses the object MBRs r_1, r_2 and r_3. In this example, it is assumed that each node can host up to three entries. However, in real implementations this number is significantly higher and depends on the page size and the number of dimensions.

It is evident that the R-tree for a collection of object is not unique. The shape of the tree depends significantly on the insertion order. As it has been pointed out previously, there are two ways to build an R-tree: (i) by individual insertion of objects and (ii) by bulk loading. In dynamic data sets, where objects may be



R-Tree (and Family). Figure 3. R-tree example.

inserted and deleted in an ad hoc manner, the first method is applied. In the sequel, the insertion process is discussed briefly.

Insertions in an R-tree are handled similarly to insertions in a B^+-tree. In particular, the R-tree is traversed to locate an appropriate leaf node L to accommodate the new entry. The selection of L involves a number of internal node selections towards determining an appropriate path from the root to the most convenient leaf. It is important to guarantee that after the insertion, the tree will be in a good shape. Towards this goal, the insertion process tries to select the next node in the insertion path, aiming at low MBR enlargement, because the size of MBRs is directly connected to search efficiency. The new entry is inserted in L and then all nodes within the path from the root to L are updated accordingly (adjustment of MBRs). In case the found leaf cannot accommodate the new entry because it is full (it already contains M entries), then it is split into two nodes. Splitting in R-trees is different from that of the B^+-tree, because it considers different criteria. Guttman in his original paper [6] proposed three different split policies:

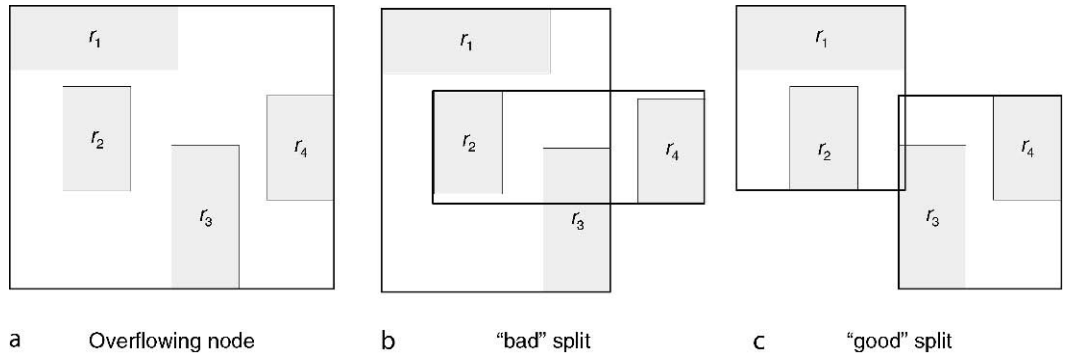
Linear Split. Choose two objects as seeds for the two nodes, where these objects are as far apart as possible. Then consider each remaining object in a random order and assign it to the node requiring the smallest enlargement of its respective MBR.

Quadratic Split. Choose two objects as seeds for the two nodes, where these objects if put together create as much dead space as possible (*dead space* is the space that remains from the MBR if the areas of the two objects are ignored). Then, until there are no remaining objects, insert the object for which the difference of dead space if assigned to each of the two nodes is maximized in the node that requires less enlargement of its respective MBR.

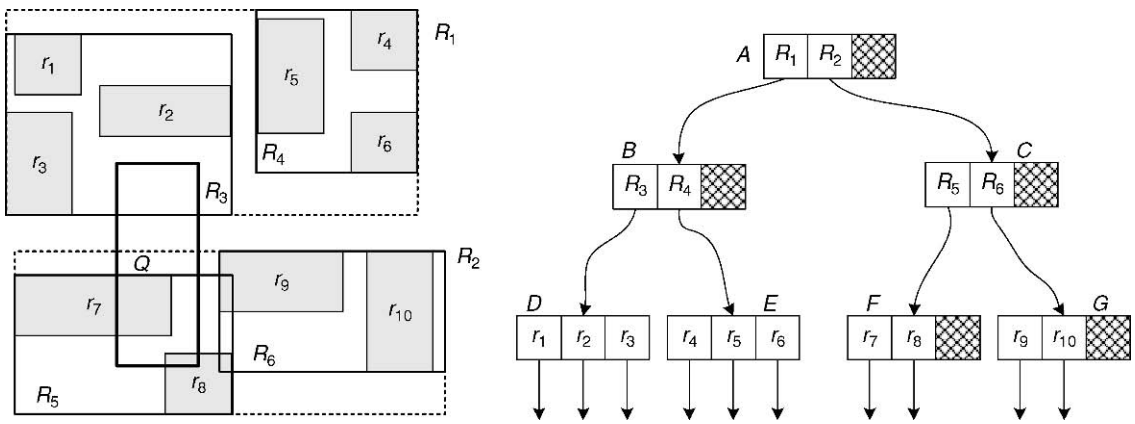
Exponential Split. All possible groupings are exhaustively tested and the best is chosen with respect to the minimization of the MBR enlargement.

Guttman suggested using the quadratic algorithm as a good compromise between insertion speed and retrieval performance. The linear split policy is the most efficient but the resulting R-tree does not keep its nice characteristics, whereas the exponential split policy although it takes the best split decision it requires significant computational overhead. Albeit the split policy being used, splits may propagate upwards up to the root node. If there is a split in the root, a new root node is created and the height of the tree increases. To demonstrate the importance of the splitting policy an example is given in Fig. 4. Figure 4a depicts an overflowing node, assuming that at most three entries can be stored. A "bad" split is shown in Fig. 4b whereas Fig. 4c depicts a "good" split choice. The first split should be avoided since the quality of the resulting nodes degrades (large MBRs with a lot of overlap), whereas the second is more preferable (small MBRs with small overlap).

Deletions are performed by first searching the tree to locate the corresponding leaf L which contains the deleted object. After the removal of the entry from L the node may contain fewer than m entries (node underflow). The handling of an underflowing node is different in the R-tree, compared with the case of B^+-tree. In the latter, an underflowing case is handled by merging two sibling nodes. Since B^+-trees index one-dimensional data, two sibling nodes will contain consecutive entries. However, for multi-dimensional data, this property does not hold. Although one still



R-Tree (and Family). Figure 4. Splitting example.



R-Tree (and Family). Figure 5. Range query example using an R-tree.

may consider the merging of two R-tree nodes that are stored at the same level, reinsertion is more appealing for the following reasons:

1. Reinsertion achieves the same result as merging. Additionally, the algorithm for insertion is used. Moreover, the pages required during reinsertion are likely to be available in the buffer memory, because they have been retrieved during the search of the deleted entry.
2. As described, the insertion process tries to maintain the good quality of the tree during the query operations. Therefore, it sounds reasonable to use reinsertion, because the quality of the tree may degrade after several deletions.

In all R-tree variants that have appeared in the literature, tree traversals for any kind of operations are executed in a way similar to the one applied in the original R-tree. An exception is the R^+-tree which decomposes an object to smaller parts and therefore, the search process requires some modifications in comparison to other variants. Basically, the dynamic variations of R-trees differ in how they perform splits and how they handle insertions in general.

To demonstrate the way queries are executed, a simple range query example is given. Figure 5 shows the region of interest Q and the query asks for all objects that intersect Q. The nodes of the R-tree are labeled using the symbols A through G. The first accessed node is A (the root). The entries of A are tested for intersection with Q. Evidently, Q intersects both R_1 and R_2, therefore both subtrees require further investigation. The next accessed node is B which contains the MBRs R_3 and R_4. Among them, only R_3 intersects the region of interest, which means that node D should be accessed next. Node D contains the object MBRs r_1, r_2 and r_3 and none of them intersects Q. At this point, the search process backtracks to node B and since no eligible entries are found it backtracks to node A. Next, node C is accessed which contains the

MBRs R_5 and R_6. Only R_5 intersects the region of interest, and therefore node F is accessed next. By inspecting the entries of F it is evident that both object MBRs r_7 and r_8 intersect Q, and both are included in the final answer. The search process backtracks to node C and since no more promising branches can be followed, it backtracks to node A. At this point, all promising branches have been examined and the range query execution terminates. The object MBRs that intersect Q are r_7 and r_8. Note that if r_7 and r_8 correspond to the real objects then no further actions should be taken. Otherwise, the detailed geometry of these objects must be tested for intersection with Q (refinement).

In the previous lines, the main issues related to R-tree construction and search have been briefly discussed. The interested reader is directed to [11] for an exhaustive list of algorithmic techniques regarding R-trees and related structures.

Key Applications

Geographic Information Systems

R-tree is an excellent choice for indexing spatial data sets. Algorithms have been proposed to answer fundamental query types like *range queries*, *nearest-neighbor queries* and more complex queries like *spatial joins* and *closest pairs* using R-trees to index the underlying data. Therefore, the structure is equipped with all necessary tools to organize and index geographic information.

Location-Based Services

Many location-aware algorithmic techniques are based on the R-tree index or its enhancements. Queries involving the current location of moving objects are handled efficiently by R-tree variants and therefore these schemes are a convenient tool for organizing objects that potentially change their location.

Multimedia Database Systems

Since the R-tree index is capable of organizing multi-dimensional data, it can be utilized as an indexing scheme for multimedia data (e.g., images, audio). A common approach to organize multimedia data is to represent the complex multimedia information by using feature vectors. These feature vectors can be organized by means of R-trees, to facilitate similarity search towards multimedia retrieval by content [5].

Future Directions

R-trees have been successfully applied to offer efficient indexing support in diverse disciplines such as query processing in spatial and spatio-temporal databases, multi-attribute data indexing, preference query processing to name a few. An important research direction towards more efficient query processing in modern systems is to provide efficient indexing schemes towards distributed processing. A significant effort towards this direction has been reported in [12], which proposes a distributed R-tree indexing scheme. Taking into account that huge volumes of multi-attribute data are scattered across different systems, such distributed schemes are expected to offer enormous help towards efficient query processing. The challenge is to provide efficient implementations of the corresponding centralized algorithms developed so far, since the methods used for centralized structures are likely to fail when applied to distributed data.

Experimental Results

The interested reader will find a plethora of performance evaluation results in the corresponding literature. Usually, when a new indexing scheme is proposed it is experimentally compared to other methods. Although these comparisons are not based on a common framework, they reveal the advantages and disadvantages of the indexing schemes under study. Usually, the comparison is based on the number of disk accesses and the computational time required to process queries.

Data Sets

Performance evaluation results regarding R-trees and related indexing schemes are produced based on real-life as well as on synthetically generated data sets with diverse distributions. The interested reader can browse the data sets hosted in http://www.rtreeportal.org to view some representative real-life data sets that are consistently being used by researchers for comparison purposes. Synthetically generated data sets follow different distributions (e.g., uniform, normal, Zipf) and their use provide additional hints for the index performance. Moreover, the use of synthetically generated data sets offers the flexibility to choose the cardinality of the data set, the distribution and the size of the objects (e.g., extent of MBRs). Hence, the comparison among indexing schemes is more reliable.

URL to Code

R-tree portal (http://www.rtreeportal.org) contains the code for most common spatial access methods (mainly R-tree and variations), as well as data generators and several useful links for researchers and practitioners interested in spatial database issues.

Cross-references

- ▶ Closest Pair Query
- ▶ Nearest-Neighbor Query
- ▶ Range Query
- ▶ Spatial Join

Recommended Reading

1. Beckmann N., Kriegel H.P., Seeger B. The R.*-tree: an efficient and robust method for points and rectangles. In ACM SIGMOD Conf. on Management of Data, 1990, pp. 322–331.
2. Berchtold S., Keim D.A., and Kriegel H.P. The X-tree: an index structure for high-dimensional data. In Proc. 22th Int. Conf. on Very Large Data Bases, 1996, pp. 28–39.
3. Chen L., Choubey R., and Rundensteiner E.A. Bulk-Insertions into R-trees using the small-tree-large-tree approach. In Proc. 6th Int. Symp. on Advances in Geographic Inf. Syst., 1998, pp. 161–162.
4. Choubey R., Chen L., and Rundensteiner E.A. GBI – A generalized R-tree bulk-insertion strategy. In Proc. 6th Int. Symp. Advances in Spatial Databases, 1999, pp. 91–108.
5. Faloutsos C. Searching Multimedia Databases by Content. Kluwer, Dordecht, 1996.
6. Guttman A. R-trees: a dynamic index structure for spatial searching. In ACM SIGMOD Conf. on Management of Data, 1984, pp. 47–57.
7. Kamel I. and Faloutsos C. On Packing R-trees. In ACM Int. Conf. on Information and Knowledge Management, 1993, pp. 490–499.
8. Kamel I. and Faloutsos C. Hilbert R-tree – an Improved R-tree using fractals. In Proc. 20th Int. Conf. on Very Large Data Bases, 1994, pp. 500–509.
9. Leutenegger S., Edgington J.M., and Lopez M.A. STR – A Simple and Efficient Algorithm for R-tree Packing. In Proc. 13th Int. Conf. on Data Engineering, 1997, pp. 497–506.
10. Lin K., Jagadish H.V., and Faloutsos C. The TV-Tree: An index structure for high-dimensional data. VLDB J. 3, 1994, 517–542.
11. Manolopoulos Y., Nanopoulos A., Papadopoulos A.N., and Theodoridis Y. R-trees: Theory and Applications. Springer, Berlin Heidelberg New York, 2006.
12. du Mouza C., Litwin W., and Rigaux P. SD-Rtree: A Scalable Distributed R-tree. In Proc. 23rd Int. Conf. on Data Engineering, 2007, pp. 296–305.
13. Roussopoulos N. and Leifker D. Direct spatial search on pictorial databases using packed R-trees. ACM SIGMOD Rec. 14 (4):17–31, 1985.
14. Sakurai Y., Yoshikawa M., Uemura S., and Kojima H. Spatial indexing of high-dimensional data based on relative approximation. VLDB J. 11(2):93–108, 2002.
15. Sellis T., Roussopoulos N., Faloutsos C. The R+-tree: a dynamic index for multidimensional objects. In Proc. 13th Int. Conf. on Very Large Data Bases, 1987, pp. 507–518.

Rule Bases

▶ Closed Itemset Mining and Nonredundant Association Rule Mining

Rule-based Classification

Anthony K. H. Tung
National University of Singapore, Singapore, Singapore

Definition

The term rule-based classification can be used to refer to any classification scheme that make use of IF-THEN rules for class prediction. Rule-based classification schemes typically consist of the following components:

- *Rule Induction Algorithm* This refers to the process of extracting relevant IF-THEN rules from the data which can be done directly using sequential covering algorithms [1,2,5–7,9,12,14–16] or indirectly from other data mining methods like decision tree building [11,13] or association rule mining [3,4,8,10].
- *Rule Ranking Measures* This refers to some values that are used to measure the usefulness of a rule in providing accurate prediction. Rule ranking measures are often used in the rule induction algorithm to prune off unnecessary rules and improve efficiency. They are also used in the class prediction algorithm to give a ranking to the rules which will be then be utilized to predict the class of new cases.
- *Class Prediction Algorithm* Given a new record with unknown class, the class prediction algorithm will predict the class of the new record based on the IF-THEN rules that are output by the rule induction algorithm. In many cases where multiple rules could be matched by the new case, the rule ranking measures will be used to either select the best best matching rule based on the ranking or to compute an aggregate from the multiple matching rules in order to arrive at a final prediction.

Historical Background

Earlier rule-based classification methods includes AQ [7], CN2 [1] and more recently RIPPER [2]. These methods induce rules using the sequential covering algorithm where. Rules are learned one at a time. Decision tree classification methods like C4.5 [13] can also be considered as a form of rule-based classification. However, decision tree induction involved parallel rule induction, where rules are induced at the same time. Even more recently, advances in association rule mining had made it possible to mine association rules efficiently in order to build a classifier [3,4,8,10]]. Such an approach can also be considered as rule-based classification.

Foundations

The discussion first looks at how IF-THEN rules can be used for classification before proceeding to look at their ranking measures and induction algorithms.

1. Using IF-THEN Rules for Classification

An IF-THEN rule is typically an expression of the form $LHS \Rightarrow RHS$ where LHS is a set of conditions that much be meet in order to derive a conclusion represented by RHS. In much literature, LHS is called the *antecedent* of the rule and RHS is called the *consequent* \of the rule. For rule-based classification, the rule antecedent typically consists of a conjunction(AND)of attribute tests while the rule consequent is a class value. An example of a IF-THEN rule will be:

$$(\text{No. of years} \geq 6) \text{ and } (\text{Rank} = \textit{Associate Professor}) \Rightarrow (\text{Tenured} = \text{YES})$$

A rule is say to *cover* a record if the record matches all the antecedent conditions in the rule. Given a new record with unknown class value, rules which cover the record will be used to determine the class value of the record. In the case where only one rule matches the tuple, the class value at the consequent of such a rule will be assigned as the predicted class for the tuple. On the other hand, it is also possible for none of the rules to match, in which case a *default class value* will be assigned.

For more complex situation in which multiple rules are matched, there are usually two approaches:

(i) Top Rule Approach In this approach, all the rules that matched the new record are ranked based on the rule ranking measures. The consequent of the rule that is rank top based on this approach will be the predicted class value of the record.

(ii) Aggregation Approach In the aggregation approach, rules that match the new record are separated into groups based on their consequent. For each of the rule group with the same consequent, an aggregated measure will be computed based on the rule ranking measure for each rule in the group. Each group of rules are then ranked based on their aggregated measure and the consequent of the rule group that are ranked highest will be the predicted class value for the new record.

Note that the two approaches described above are not mutually exclusive. For example it is possible to pick the top-k rules based on the first approach and then apply the aggregation approach on the top-k rules so as to determine the final predicted class value.

2. Rule Ranking Measures

Rule ranking measures are important components in a rule-based classification scheme because of two reasons. First, they can improve the efficiency of constructing and using the classifier. Rules that are not deemed to be useful based on these measures can be pruned off during rule induction making the process more efficient and also reducing the number of rules that must be processed during classification. Second, they can enhance the effectiveness of a rule-based classifier by removing rules which have weak prediction power.

Among the various rule ranking measure, the most basic ones are *coverage*, *accuracy* and *length of the rule*. Let n be the number of record in the training database, n_r be the number of records that match both the antecedent and consequent of a rule r and n_{lr} be the number of records that are covered by r. The coverage of a rule r, denotes as $cov(r)$ is defined to be n_{lr}/n while the accuracy of rule r is defined to be n_r/n_{lr} and denote as $acc(r)$. A rule should have high coverage and accuracy in order to be useful for classification. Besides coverage and accuracy, the length of a rule, $len(r)$, which is the number of terms at the antecedent of r, is also important in determining the usefulness of a rule in a rule-based classification scheme.

More complex rule ranking measures try to integrate both coverage and accuracy. They include *information gain* and *likelihood ratio*. Information gain was proposed in FOIL (First Order Inductive Learner) for comparing rules whose antecedents are subset/superset of each other. Let r' be a rule which

Algorithm 1 Generic Sequential Covering Algorithm

```
Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.
Input: D, a database of class-labelled records; Attvals, the set of all attributes and
their possible values.
Output: A set of IF-THEN rules. Method:
    Rule set = { };
    for each class c do
      repeat
        Rule = Learn OneRule (D, Att vals, c);
        remove tuples covered by Rule from D;
      until terminating condition;
      Rule set = Rule set + Rule; // add new rule to rule set
    return Rule Set;
```

antecedent is a superset of r. FOIL assess the information gain of r' over r to be

$$FOIL\_Gain = p' \times \left(\log_2 \frac{p'}{p'+n'} - \log_2 \frac{p}{p+n} \right) \quad (1)$$

where p', n', p' and n are the number of records that are of positive class and negative class and which are covered by r' and r respectively. FOIL_gain favors rules that have high accuracy and cover many positive tuples.

Besides, information gain, one can also use a statistical test of significance to determine if the apparent effect of a rule is not attributed to chance but instead indicates a genuine correlation between attribute values and classes. The test compares the observed distribution among classes of tuples covered by a rule with the expected distribution that would result if the rule made predictions at random. Given m classes, let p_i be the probability distribution of class i within the database and $p_i(r)$ be the probability distribution of class i within the set of records that are covered by r. The likelihood ratio is computed as:

$$Likelihood\_Ratio = 2\sum_{1}^{m} p_i(r) \log\left(\frac{p_i(r)}{p_i}\right) \quad (2)$$

The likelihood ratio is then used to perform a significant test against a χ^2 distribution with m-1 degrees of freedom. The higher the likelihood ratio is, the more likely that there is a significant difference in the number of correct predictions made by the rule in comparison with a random guess that following the class probability distribution of the database.

3. Rule Induction

Various forms of rule induction can be performed for rule-based classification. Here, the *sequential covering* algorithm will be described.

Algorithm 1 presents a generic algorithm for sequential covering rule induction. The algorithm iteratively learn one rule from the database and then remove all records that are covered by the rule before learning the next rule. This is done repeatedly until a terminating condition is met. The terminating condition can varies across different algorithms but is typically linked to the fact no more interesting rules can be induced once many of the records are removed from the database. All rules that are induced during the process are then output.

Each single invocation of LearnOneRule typically involve a greedy search for interesting rule based on the rule ranking measures. This is done by searching for attribute conditions which will improve the rule ranking measure when they are appended to the antecedent of the rule. A single attribute condition that best improve the measure is appended each time and this is done repeatedly until the measure cannot be improved. The rule will then be return as the output for LearnOneRule.

Key Applications

Rule-based classification has been very popularly used in machine learning for classification of data. It is applicable whenever other classification scheme is applicable.

Future Directions

Despite efforts to reduce the number of rules in a rule-based classifier, the number of rules being used are still substantially higher than what a human can handle. More studies on new interestingness measure and visualization techniques are needed to further enhance the interpretability of a rule-based classifier.

Cross-references

- ▶ Associative Classifiers
- ▶ Association Rule Mining
- ▶ Decision Tree Induction
- ▶ Sequential Covering Algorithm BBb

Recommended Reading

1. Clark P. and Niblett T. The CN2 induction algorithm. Mach. Learn., 3(4):261–283, 1989.
2. Cohen W. Fast effective rule induction. In Proc. 12th Int. Conf. on Machine Learning, 1995, pp. 115–123.
3. Cong G., Tan K., Tung A., and Xu X. Mining top-K covering rule groups for gene expression data. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2005, pp. 670–681.
4. Cong G. Tung A.K.H., Xu X., Pan F., and Yang J. FARMER: finding interesting rule groups in microarray datasets. In Proc. ACM SIGMOD Int. Conf. on Management of Data, 2004, pp. 143–154.
5. Domingos P. The RISE system: conquering without separating. Tools with Artificial Intelligence, 1994. In Proc. 6th IEEE Int. Conf. on Tools with Artificial Intelligence, 1994, pp. 704–707.
6. Furnkranz J. and Widmer G. Incremental reduced error pruning. In Proc. 11th Int. Conf. on Machine Learning, 1994, pp. 70–77.
7. Hong J., Mozetic I., and Michalski R. AQ15: Incremental Learning of Attribute-Based Descriptions from Examples: The Method and User's Guide. Reports of the Intelligent Systems Group, ISG, pp. 86–5.
8. Liu B., Hsu W., and Ma Y. Integrating Classification and Association Rule Mining. In Proc. 4th Int. Conf. on Knowledge Discovery and Data Mining, 1998.
9. Major J. and Mangano J. Selecting among rules induced from a hurricane database. J. Intell. Inform. Syst., 4(1):39–52, 1995.
10. Pan F., Cong G., and Tung A.K.H. CARPENTER: Finding closed patterns in long biological datasets. In Proc. 9th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, 2003.
11. Quinlan J. Simplifying decision trees. Int. J. Man–Machine Studies, 27(3):221–234, 1987.
12. Quinlan J. Learning logical definitions from relations. Mach. Learn., 5(3):239–266, 1990.
13. Quinlan J. C4.5: Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA, 1993.
14. Quinlan J. and Cameron-Jones R. FOIL: A Midterm Report. In Proc. European Conf. on Machine Learning, 1993.
15. Smyth P. and Goodman R. An information theoretic approach to rule induction from databases. IEEE Trans. Knowl. Data Eng., 4(4):301–316, 1992.
16. Weiss S. and Indurkhya N. Predictive Data Mining: A Practical Guide. Morgan Kaufmann, Los Altos, CA, 1998.