Recommended Reading

1. Akkiraju R, Farrell J, Miller J, et al. Web service semantics – WSDL-S, vol. 1.0, tech. note, Apr. 2005. At http://www.w3.org/Submission/WSDL-S/.
2. Battle S, Bernstein A, Boley H, et al. Semantic Web Services Framework (SWSF) Overview, 2005. At http://www.w3.org/Submission/SWSF/.
3. Cabral L, Domingue J, Galizia S, et al. IRS-III: a broker for semantic web services based applications. In: Proceedings of the 5th International Semantic Web Conference; 2006. p. 201–14.
4. Fensel D, Lausen H, Polleres A, et al. Enabling semantic Web services: the Web service modeling ontology. New York: Springer; 2006.
5. Martin D, Burstein M, McDermott D, et al. Bringing semantics to web services with OWL-S. World Wide Web J. 2007;10(3):243–77.

Semantics-Based Concurrency Control

Krithi Ramamritham[1] and Panos K. Chrysanthis[2]
[1]Department of Computer Science and Engineering, Indian Institute of Technology Bombay, Mumbai, India
[2]Department of Computer Science, University of Pittsburgh, Pittsburgh, PA, USA

Definition

Specifications of data contain semantic information that can be exploited to increase concurrency. For example, two insert operations on a multiset object commute and hence, can be executed in parallel; further, regardless of whether one operation commits, the other can still commit. Applying the same rule, two push operations on a stack object do not commute and hence cannot be executed concurrently. Several schemes have been proposed for exploiting the semantics of operations have to provide more concurrency than obtained by the conventional classification of operations as *reads* or *writes*.

Key Points

In most semantics-based protocols, conflicts between operations is based on commutativity, an operation o_i which does not commute with other uncommitted operations will be made to wait until these conflicting operations abort or commit. Some protocols use operations' return value commutativity, wherein information about the results of executing an operation is used in determining commutativity, and some use the arguments of the operations in determining whether or not two operations commute. An example of the former, two increment operations on a counter object commute as long as they do not return the new or old value of the counter. An example of the latter, two insert operations on a set object commute as long as they do not insert the same item.

In the scheme reported in [1], non-commuting but *recoverable* operations are allowed to execute in parallel; but the order in which the transactions invoking the operations should commit is fixed to be the order in which they are invoked. If o_j is executed after o_i, and o_j is *recoverable relative to* o_i , then, if transactions T_i and T_j that invoked o_i and o_j respectively commit, T_i should commit before T_j . Thus, based on the recoverability relationship of an operation with other operations, a transaction invoking the operation sets up a dynamic commit dependency relation between itself and other transactions. If an invoked operation is not recoverable with respect to an uncommitted operation, then the invoking transaction is made to wait. For example, two pushes on a stack do not commute, but if the push operations are forced to commit in the order they were invoked, then the execution of the two push operations is serializable in commit order. Further, if either of the transactions aborts the other can still commit.

In [2] authors make an effort to discover, from first principles, the nature of concurrency semantics inherent in objects. Towards this end, they identify the dimensions along which object and operation semantics can be modeled. These dimensions are then used to classify and unify existing semantic-based concurrency control schemes. To formalize this classification, a graph representation for objects that can be derived from the abstract specification of an object is proposed. Based on this representation, which helps to identify the semantic information inherent in an object, a methodology is presented that shows how various semantic notions applicable to concurrency control can be effectively combined to improve concurrency. A new source of semantic information, namely, the ordering among component objects, is exploited to further enhance concurrency. Lastly, the authors present a scheme, based on this methodology, for *deriving* compatibility tables for operations on objects.

Cross-References

- ▶ ACID Properties
- ▶ Concurrency Control: Traditional Approaches

Recommended Reading

1. Badrinath BR, Ramamritham K. Semantics-based concurrency control: beyond commutativity. ACM Trans Database Syst. 1991;17(1):163–99.
2. Chrysanthis PK, Raghuram S, Ramamritham K Extracting concurrency from objects: a methodology. In: Proceedings of the ACM SIGMOD International Confernce on Management of Data; 1991.

Semijoin

Kai-Uwe Sattler
Technische Universität Ilmenau, Ilmenau, Germany

Synonyms

Bit vector join; Bloom filter join; Bloom join; Hash filter join; Semijoin filter

Definition

Semijoin is a technique for processing a join between two tables that are stored at different sites. The basic idea is to reduce the transfer cost by first sending only the projected join column(s) to the other site, where it is joined with the second relation. Then, all matching tuples from the second relation are sent back to the first site to compute the final join result.

Historical Background

The semijoin technique was originally developed by Bernstein et al. [3] as part of the SDD-1 project as a reduction operator for distributed query processing. The idea of applying hash filtering was proposed by Babb [1] as well as by Valduriez [9] particularly for specialized hardware (content addressed file stores and distributed database machines respectively). The theory of semijoin-based distributed query processing was presented in [2]. In [10] semijoins are also exploited for query processing on multiprocessor database machines. Results of detailed experimental work on semijoins in distributed databases were first reported by Lu and Carey [6] as well as by Mackert and Lohman [7].

Foundations

Semijoin is a join processing technique which was originally developed for distributed databases. A semijoin is the "half of a join" and is particularly useful as a reduction operator.

Relational Definition

Given two relations $R(A,B)$ and $S(C,D)$ with the join condition $R.A = S.C$ the semijoin R ⋉ S is defined as follows:

$$R \ltimes_{A=C} S = \pi_{attr(R)}(R \bowtie_{A=C} S)$$

where $attr(R)$ denotes the set of attributes in R. The semijoin has two important characteristics: