Semipermeable Transactions and Semantics-Based Concurrency Control for Multidatabases*

John Shillington
M. Tamer Özsu
Laboratory for Database Systems Research
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1
{john,ozsu}@cs.ualberta.ca

Abstract

The problem of concurrency control for transactions in a multidatabase system has received considerable attention over the past few years. Many of the proposed solutions offer either small increases in concurrency compared to sequential execution, or make unrealistic assumptions about the operating environment. We approach the problem by combining existing solutions which guarantee global serializability and then use the semantics of global transactions to significantly increase the degree of concurrency. The idea is to specify acceptable violations of global serializability when allowed by the semantics of global transactions. This is accomplished by allowing global transactions to interleave globally based on what we call the permeability *of the transactions.*

1 Introduction

Our work deals with (a) the specification of a new transaction model called *semipermeable transactions* which are nested in their structure, (b) the development of a semantics-based correctness criterion based on previous work involving centralized database management systems (DBMSs) [1], and (c) the development and implementation of concurrency control algorithms that enforce this criterion. This paper summarizes our current thinking and indicates the direction of the research.

The paper is organized as follows. In Section 2 we present the multidatabase system (MDBS) model that we assume in this study. In Section 3 we motivate the development of the semipermeable transaction model and present it briefly. Section 4 provides an overview of the semantics-based correctness criterion that we are using and Section 5 outlines the concurrency control algorithms that may be used. Finally Section 6 places this paper within the general context of our research.

*This research has been supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) under grant OGP0951 and by travel assistance provided to the first author by the Mary Louise Imrie Graduae Award, Faculty of Graduate Studies and Research, and the Vice-President (Research) of the University of Alberta.

2 MDBS Model

In this paper we consider a model in which each component DBMS of a multidatabase system has its own transaction manager (called *local transaction manager*) and the MDMS layer on top has its own transaction manager (called the *global transaction manager*). Accordingly, there are two types of transactions: *local transactions*, which are submitted to each DBMS, and *global transactions*, which are submitted to the MDMS layer. Local transactions execute on a single database, whereas global transactions access multiple databases. To simplify our initial research, we require that a global transaction be divided into a set of *global subtransactions*, each of which executes on one database; the subtransactions must be executed in a predefined sequence. The global subtransactions within a global transaction are called *sibling subtransactions*, and the global transaction to which they belong is called the *parent transaction*. Global subtransactions from different global transactions which operate on the same component database are called *corresponding subtransactions*. A global subtransaction may therefore depend on values derived in previously executed sibling subtransactions, and may derive values which are used in following sibling subtransactions.

3 Semipermeable Transactions

Our interest in relaxed transaction models and semantics-based correctness criteria is motivated by the observation that the level of concurrency provided by flat transactions that are scheduled according to serializability is typically relatively low. Consider using a semantics-based correctness criterion at the global level. If global transactions can be categorized and allowable misorderings (even if they are

illegal under serializability) of global subtransactions specified, a higher rate of concurrency can be achieved than under global serializability. In the best case, if the semantics of the global transactions in a particular multidatabase environment allow arbitrary misorderings of global transactions, all global transactions can be submitted concurrently without any fear of later rejection due to misorderings. Realistically, it is likely that global transactions will range from *impermeable transactions* (no misorderings allowed) to *completely permeable transactions* (arbitrary misorderings allowed) for any given multidatabase environment. Between impermeable and completely permeable transactions, there are *semipermeable transactions*, which allow some misorderings between transactions.

The transactions in this model are called semipermeable because of the analogy to semipermeable membranes in biology, which selectively let some molecules pass through the membrane while keeping others out. The idea is that transactions in this model may selectively interleave, or permeate each other, based on semantic information about the contents of the transactions. Because of this selective interleaving, they may be thought of as being semipermeable. The degree to which transactions may permeate each other determines the amount of concurrency that may be achieved. The permeability of transactions is determined based on the correctness criterion for concurrent execution, which we will discuss in the next section.

In our research, semipermeable transactions are used to model global transactions, without any restriction on the local ones. This is consistent with the autonomy of component DBMSs which have their own transaction models. The definition borrows liberally from [1], but we extend it to apply to the multidatabase environment. A semipermeable (global) transaction GT_i is modeled as a sequence of subtransactions and a termination command ($TR(T_i)$) at the end of this sequence: $GT_i = GST_{i1}GST_{i2}\ldots GST_{in}TR(GT_i)$. Each global subtransaction GST_{ij} consists of a sequence of atomic actions on objects at one component database. An action is either a read operation or a write operation on a database object x. A transaction can read and write an object once and if both occur, read must precede write and both must occur within the same subtransaction. Therefore, each subtransaction encapsulates actions that are performed on one of the component databases. The termination command $TR(GT_i)$ represents the termination point of all of the subtransactions of GT_i (commit or abort).

Example 1 Consider an MDBS consisting of two component databases with the following data objects: $LDB_1 = \{a, b, c\}$, $LDB_2 = \{x, y, z\}$. Assume two global transactions are posed on the system (where r, w, c stand for *read*, *write*, and *commit*, respectively):

$$GT_1 : r(a), w(a), r(x), w(y), c$$

$$GT_2 : w(b), w(c), r(z), w(z), c$$

These global transactions can be modeled as follows:

$$GT_1 = \underbrace{r_{11}(a)\ w_{11}(a)}_{GST_{11}}\ \underbrace{r_{12}(x)\ w_{12}(y)}_{GST_{12}}\ TR(GT_1)$$

$$GT_2 = \underbrace{w_{21}(b)\ w_{21}(c)}_{GST_{21}}\ \underbrace{r_{22}(z)\ w_{22}(z)}_{GST_{22}}\ TR(GT_2)$$

□

To capture the relevant semantic information about global transactions, we use *interleaving sets*. An interleaving set specifies which global transactions may interleave after a particular global subtransaction. The elements of an interleaving set therefore specify acceptable violations of global serializability. Associated with each subtransaction GST_{ij} in a global transaction GT_i is an *interleaving set* $IS(GST_{ij})$, which denotes the global transactions that are allowed to interleave after GST_{ij}[1] This means that if $GT_k \in IS(GST_{ij})$, then all of the subtransactions of global transaction GT_k can be scheduled to execute following GST_{ij} even if this violates global serializability.

Example 2 Suppose the interleaving sets for the global transactions of the previous example are specified as follows:

$$IS(GST_{11}) = \{GT_2\}$$

$$IS(GST_{21}) = \{\}$$

This effectively specifies that, besides the globally serializable schedules, one schedule involving the two transactions that is not globally serializable is also allowed: $GST_{11}, GST_{21}, GST_{22}, GST_{12}$ □.

Whether a transaction is included in an interleaving set of a subtransaction or not is determined strictly according to the semantics of the two transactions.

4 Semantics-Based Correctness

We need a correctness criterion which will enable us to determine the allowable interleavings of semipermeable transactions. The correctness criterion that we propose to use is *relative consistency* which is defined in detail in [1]. Briefly, a schedule H is considered to be correct if sibling subtransactions occur in their predefined sequence, and any interleaving among subtransactions is acceptable according to the semantics of the transactions as specified by the interleaving sets. We now present a slightly more detailed definition.

An operation in GST_{ij} is said to be in *direct conflict* with another operation in GST_{kj} ($i \neq k$) iff both operations access the same object and at least one of them is a write operation. An operation in GST_{ij} is

[1] The last subtransaction in a global transaction does not require an interleaving set since there are no further sibling subtransactions for another global transaction to interleave with. We include it only when needed for uniformity.

said to be in *indirect conflict* with another operation in GST_{kj} iff there are a sequence of direct conflicts leading from GST_{ij} to GST_{kj} through one or more other transactions (either local or global). As a result of the autonomy of component databases, the global scheduler cannot determine whether or not indirect conflicts exist between corresponding subtransactions. Therefore, two subtransactions can conflict only if they execute on the same component database, and all corresponding subtransactions must be assumed to conflict because of the possibility of indirect conflicts. A schedule H is considered to be correct if two conditions hold:

1. H is stepwise serial [3], meaning that the operations of sibling subtransactions appear in their correct order in H. In other words, the operations of each subtransaction appear in their correct order in H, and the operations from sibling subtransactions are not interleaved with each other.

2. For every successive pair of sibling subtransactions (say GST_{ij} and GST_{ij+1}) and for every different subtransaction GST_{kl} occurring between GST_{ij} and GST_{ij+1} it must be the case that $GST_{kl} \in IS(GST_{ij})$. This ensures that any interleaving among the subtransactions is acceptable according to the semantics of the transactions.

It is possible to construct a *precedence graph* for a schedule H where the nodes consist of all the subtransactions of global transactions and directed arcs are added as follows:

1. For each pair of consecutive sibling subtransactions, a directed arc exists from the preceding to the succeeding subtransaction (called an *internal arc*).

2. For each pair of corresponding subtransactions, there is a directed arc from the preceding subtransaction to the following subtransaction (called a *conflict arc*)[2]. Call the preceding subtransaction the *conflict tail* and the following subtransaction the *conflict head*.

3. If the parent of a conflict head is not specified in the interleaving set of the conflict tail, a directed arc exists which terminates on the conflict head and emanates from the first sibling following the conflict tail which *does* allow the parent of the conflict head to interleave (called an *interleaving arc*). This will either be the last sibling of the conflict tail (if no interleaving is allowed), or the first sibling after the conflict tail which contains the parent of the conflict head in its interleaving set.

One can then define a *relatively consistent* schedule as one whose precedence graph is acyclic. Because of the nature of the MDBS model that we are using, the acyclicity of the precedence graph is a sufficient condition for guaranteeing that there exists a topological sort of the precedence graph which conforms to our definition of a correct schedule. This is in contrast to the results presented in [1] for a centralized database, which require not only acyclicity of the precedence graph, but verification that a "correct" topological sort exists.

[2]Note that because of the possibility of indirect conflicts we must assume that a conflict exists between each pair of corresponding subtransactions.

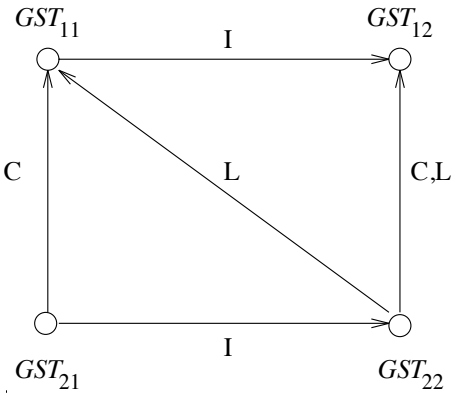

Figure 1: Precedence Graph

Example 3 Consider the global transactions of the previous examples. Suppose that the conflict relations are such that $GST_{21} \rightarrow GST_{11}$ and $GST_{22} \rightarrow GST_{12}$, and the interleaving sets are specified as in Example 2. The precedence graph is shown in Figure 1, with internal arcs denoted by I, conflict arcs by C, and interleaving arcs by L. □

5 Concurrency Control Algorithm

We now outline a way of incorporating semipermeable transactions and semantics-based concurrency control into an optimistic scheduling algorithm. The scheme that we propose is simple to manage, yet takes advantage of any specified permeability, allowing degrees of increased multidatabase transaction concurrency.

Consider an optimistic scheduling approach, as typified by the *Optimistic Ticket Method (OTM)* [2]. OTM allows multiple global transactions to be submitted concurrently by forcing direct conflicts among all global transactions at each component database. By forcing conflicts on a particular data object (called a *ticket*) at each component database, the local serialization order of global subtransactions can be determined by the global scheduler. The global scheduler uses this information to determine if global serializability has been violated, and if so, aborts and resubmits subtransactions as necessary in order to maintain global serializability.

To understand how to modify OTM when relative consistency is used as the correctness criterion, we need to consider how to test for relative consistency. Instead of using a *Global Serialization Graph* test to validate the schedule, as described in [2], we

substitute a *Relative Consistency Precedence Graph* test. In a global serialization graph, nodes represent global transactions and arcs represent conflicts between global transactions. In contrast, nodes in an RC precedence graph represent global subtransactions and arcs are of the three types described above (i.e. internal arcs, conflict arcs, and interleaving arcs). Briefly, the global scheduler works as follows: When a new global transaction arrives, its first subtransaction is submitted to the appropriate component DBMS. When the subtransaction is prepared to commit, a node representing that subtransaction is added to the RC precedence graph. Conflict arcs are drawn between the new node (subtransaction) and any corresponding subtransactions based on their relative ticket values: a conflict arc emanates from the node with the lesser ticket value and terminates on the node with the greater ticket value. Interleaving arcs are then added as follows: (1) all preceding siblings of the new node are checked to determine if they require an interleaving arc to emanate from the new node, and if so an arc is added; (2) for each conflict arc incident on the new node, an interleaving arc is also added (when possible) by checking the appropriate interleaving sets. As each new arc is added, the graph is checked for a cycle. If all of the arcs are added without creating a cycle, the schedule is relatively consistent, as defined above. If relative consistency is violated (i.e. a cycle exists), the node and all newly created arcs are removed from the graph. The subtransaction is then restarted and the procedure is repeated.

If relative consistency is satisfied, the next sibling subtransaction is submitted to the appropriate component DBMS. When the subtransaction is prepared to commit, a node is added to the precedence graph and an internal arc is drawn from the previous sibling subtransaction to the current subtransaction. The procedure for adding conflict arcs and interleaving arcs and checking for relative consistency is then repeated. Each remaining sibling subtransaction is submitted, executed, and checked for relative consistency in the same way. When all sibling subtransactions have been successfully added to the graph, they are committed.

The method outlined above is only one of the possible algorithms for a global scheduler that uses relative consistency as its correctness criterion. It seems likely that other concurrency control methods could be adapted to take advantage of the semantic information provided by the interleaving sets, resulting in a higher level of concurrency while still producing relatively consistent schedules. For example, a conservative approach such as the Global Serial Scheduler proposed in [4] could potentially be modified to incorporate interleaving sets to increase the degree of concurrency obtainable. Another possibility is to combine the notion of relative consistency with other relaxed forms of serializability, such as *two level serializability*[5], to increase the set of allowable schedules. It is our view that the use of semantic information should not exclude other concurrency control techniques, but rather should enhance them by increasing the number of acceptable schedules. Interleaving sets and relative consistency provide one means of doing so.

6 Conclusion

We have outlined our current research involving a nested transaction model which we call semipermeable transactions, and a semantics-based correctness criterion for controlling global transactions in a multidatabase system. We should indicate that this research is still ongoing and what is presented in this paper should not be taken as the definitive results. There are a large number of questions that we are currently investigating, including those discussed herein. Some of the other problems that we are considering are:

- the effect of removing some of the restrictions on the types of global transactions allowed in the model;
- capturing semantic information by defining classes of global transactions instead of using interleaving sets;
- automatic determination of semantic information based on the specification of global constraints;
- adaptive concurrency control techniques which consider such things as the level of autonomy of component DBMSs, concurrency control mechanisms of component DBMSs, and multidatabase transaction requirements to maximize the degree of concurrency that may be achieved in a given multidatabase environment;
- recovery issues related to our models.

This paper serves as a road map of our research and provides an initial statement of the approach that we are taking.

References

[1] A.A. Farrag and M.T. Özsu. Using semantic knowledge of transactions to increase concurrency. *ACM Transactions on Database Systems*, 14(4):503–525, December 1989.

[2] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. On serializability of multidatabase transactions through forced local conflicts. In *Proc. 7th Int. Conf. on Data Engineering*, pages 314–323, 1991.

[3] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems*, 18(2):186- -213, June 1983.

[4] K. Barker and M.T. Özsu. Transaction Management techniques for multidatabase systems. *under submission*, 1992.

[5] S. Mehrotra, R. Rastogi, H.F. Korth, and A. Silberschatz. Maintaining database consistency in heterogeneous distributed database systems. Technical Report, TR-91-04, University of Texas, February 1991.