NORTH-HOLLAND

Applications

Design and Evaluation of a New Transaction Execution Model for Multidatabase Systems

TİMUÇİN DEVİRMİŞ

and

ÖZGÜR ULUSOY*

Department of Computer Engineering and Information Science, Bilkent University, Bilkent, Ankara 06533, Turkey

Communicated by Ahmed Elmagarmid

ABSTRACT

In this paper, we present a new transaction execution model that captures the formalism and semantics of various extended transaction models and adopts them to a multidatabase system (MDBS) environment. The proposed model covers nested transactions, various dependency types among transactions, and commit independent transactions. The formulation of complex MDBS transaction types can be accomplished easily with the extended semantics captured in the model. A detailed performance model of an MDBS is employed in investigating the performance implications of the proposed transaction model. © Elsevier Science Inc. 1997

1. INTRODUCTION

A multidatabase system (MDBS) is an integrated database system that provides a global view and uniform access to different local components without requiring the users to know the individual characteristics of the

*Corresponding author who can be contacted via oulusoy@bilkent.edu.tr.

INFORMATION SCIENCES **102**, 203–238 (1997)

655 Avenue of the Americas, New York, NY 10010

0020-0255/97/$17.00
PII S0020-0255(97)00015-7

participant databases. Each local database system (LDBS) can have a different data model, and different transaction management and concurrency control mechanisms. Integration of heterogeneous components should not violate the autonomy of LDBSs, which is the most important feature of MDBSs that distinguishes them from conventional distributed database systems [5, 11, 14, 18].

Heterogeneity of the components in an MDBS leads to a requirement for flexible and powerful ways of accessing the data. The need for the coordination of the activities that belong to independent data sources makes it difficult to adopt traditional transaction control methods in an MDBS environment. Traditional transaction models generally assume a competition among transactions, but in an MDBS, sometimes cooperation besides the competition also is required for efficient processing of transactions. Defining and observing dependencies among the transactions executed over different sites can significantly affect the system performance. The variance among the execution times of transactions over different local DBMSs also forces the existing models to be reorganized accordingly. Also, the properties like atomicity and isolation introduced by the traditional transaction model are sometimes inapplicable in an MDBS environment. Under all of those considerations, we can safely argue that it is necessary to modify and extend existing distributed transaction models for MDBS environments.

In this paper, we present a new transaction model for MDBSs. This model captures the formalism and semantics of various extended transaction models, and adopts them to an MDBS environment. The extended models constituting our transaction model are the nested transactions [13], the flexible transaction model that provides various dependency relations among transactions [19], and the model that involves a relaxed version of transaction atomicity, namely the semantic atomicity, to increase the level of concurrency [6, 12]. While including the semantics of all those transaction models, the global serializability in our execution model was ensured through the use of the ticketing method [9].

In the *nested transaction* model [13], flat transactions are enhanced by a hierarchical control structure. Each nested transaction consists of either primitive transactions or some nested transactions that are called subtransactions of the containing transaction. The whole transaction structure can be represented by a tree. The root of the tree is called the top-level transaction. A transaction that contains subtransactions is a parent transaction, and the subtransactions are the children of that transaction. In the nested transaction model, a child starts after its parent, and terminates before the parent terminates. The parent is not allowed to terminate

before all of its child transactions are terminated. However, if a child is aborted, the parent does not need to be aborted.

If a distributed transaction is executed over multiple sites in the form of subtransactions, we cannot ignore the dependencies that can occur among the subtransactions [19]. A possible dependency among subtransactions is the execution order dependency in which a subtransaction cannot be executed before some others complete their executions. That kind of dependency relation is often referred to as a *precedence* relation among subtransactions. Another kind of dependency can be specified if some subtransactions are alternatives of some others. In an *alternative* dependency, one of the functionally equivalent subtransactions needs to be executed. If the user assigns priority to alternative subtransactions, a *preference* relation exists among those subtransactions.

In [12], the following types are defined for the subtransactions of a distributed transaction. A *compensatable* subtransaction can commit before its containing transaction commits, and if that transaction aborts, the effects of the subtransaction on the database can be undone by executing the associated compensating subtransaction. The *retriable* transactions are subtransactions that eventually succeed if they are retried a sufficient number of times. A retriable subtransaction can be allowed to commit later than its containing transaction. The compensatable and retriable transactions, which are also called *commit-independent* transactions, reduce the blocking effects of the commitment protocols.

In the second part of the paper, we investigate the impact of various dependency relations among transactions, and commit-independent transactions on the performance of an MDBS. A detailed simulation model of an MDBS is employed in evaluating the performance of both global and local transactions. The simulation model, which captures the basic characteristics of an MDBS, is also used in studying the comparative performance implications of two different ticketing-based concurrency control protocols. To the best of our knowledge, no performance evaluation work has appeared in the literature exploring ticketing methods or extended transaction semantics in MDBSs.

The remainder of the paper is organized in the following manner. The next section presents how the extended transaction semantics described briefly above can be combined in a transaction model for MDBSs. Section 3 provides the details of an execution architecture for the transactions of the proposed model. Section 4 describes the simulation model of an MDBS. Section 5 provides the results of the performance experiments. The last section summarizes the conclusions of our work.

2. A MULTIDATABASE TRANSACTION MODEL

Two types of transactions can coexist in an MDBS: *local transactions* and *global transactions*. A transaction that is generated and executed at the same site is called a local transaction. A global transaction, on the other hand, can submit operations to multiple sites, and at each of those sites, a *subtransaction* is executed on behalf of the global transaction.

In a multidatabase environment, some subtransactions can be committed independently of their parent transaction. If a subtransaction's effects on the database can be semantically undone by executing a compensating transaction, the subtransaction can be allowed to commit earlier. A subtransaction that reserves a seat in an airline reservation system is compensatable by a transaction that cancels the reservation. Another kind of commit-independent subtransactions is the retriable subtransactions which eventually commit if they are retried a number of times. A retriable subtransaction can be committed later than its parent transaction. Crediting a bank account is an example of retriable subtransactions. We will consider three transaction types (TT) in our model:

- Compensatable (C),
- Retriable (R), or
- Ordinary (O) (neither compensatable nor retriable).

In the following, we provide a formal definition for subtransactions processed in our MDBS and the dependency relation types among transactions.

DEFINITION 1. A subtransaction S is a 2-tuple $S = (TT, CT)$ where:

- TT is the transaction type of S,
- CT is the set of compensating transactions of S, if TT is compensatable (an empty set otherwise).

DEFINITION 2. Let T_i and T_j be two transactions.[1] We define four types of dependency relations between T_i and T_j.

- Precedence relation ($<$): $T_i < T_j$ means that T_j cannot begin execution until T_i successfully finishes its execution.
- Alternative relation ($\diamond$): $T_i \diamond T_j$ means that T_j and T_i are alternatives of each other, and any of them can be executed. It is also possible to execute them together, but only one of them should be committed.

[1] Each of T_i and T_j can be either a subtransaction or a global transaction.

- Preference relation ($\triangleright$): $T_i \triangleright T_j$ means that among two alternative transactions T_i and T_j, T_i is preferred to T_j. If they are executed together, T_j can be committed only if T_i fails. If they are not allowed to execute together, T_i should execute first, and if it fails, T_j can be executed.
- No-dependency relation ($\square$): $T_i \square T_j$ means that T_i and T_j can execute independently.

A global transaction in our model is syntactically a nested transaction with extended semantics. A global transaction consists of a set of child transactions, each of which is either a subtransaction or again a global transaction. This transaction model can be represented as a tree where the internal nodes are global transactions and the leaf nodes are subtransactions. The height of a transaction tree can vary depending on the complexity of the transaction.

DEFINITION 3. A global transaction G is a 3-tuple $G=(ST, DT, TO)$ where:

- ST is the set of global transactions and/or subtransactions that are the children of G,
- DT is the dependency type among the transactions in ST,
- TO is the total order on ST according to the dependency specified in DT.

EXAMPLE 1.[2] Consider a travel agent information system. In this system, a transaction may consist of the agent's negotiation with airlines for the flight ticket, the negotiation with car rental companies for a car reservation, and the negotiation with hotels to reserve a room. Assume for a given trip that the only airlines available are Northwest and United, the only car rental company is Hertz, and the only hotels in the destination city are Hilton, Sheraton, and Ramada. The agent can order a ticket from either Northwest or United, but Northwest is preferred; a car is mandatory for the trip; and any of the three hotels is suitable for the customer needs. Further, only the reservation of the hotel room can be canceled. The following subtransactions can be defined for a global transaction that should be executed for this application:

- S_1: Order a ticket at Northwest Airlines.
- S_2: Order a ticket at United Airlines.
- S_3: Rent a car at Hertz.

[2]Adapted from an example in [6].

- S_4: Reserve a room at Hilton.
- S_5: Reserve a room at Sheraton.
- S_6: Reserve a room at Ramada.
- S_7: Cancel a room reservation at Hilton.
- S_8: Cancel a room reservation at Sheraton.
- S_9: Cancel a room reservation at Ramada.

The global transaction G_{trip} can be specified as follows (see Figure 1):

$$
\begin{aligned}
G_{\text{trip}}(ST = \{ & G_{\text{airlines}}(ST = \{S_1(TT = O, CT = \{\})\}, \\
& \qquad S_2(TT = O, CT = \{\}), \\
& \qquad DT = \text{Preference}, \\
& \qquad TO = S_1 \triangleright S_2), \\
& S_3(TT = O, CT = \{\}), \\
& G_{\text{hotel}}(ST = \{S_4(TT = C, CT = \{S_7\}), \\
& \qquad S_5(TT = C, CT = \{S_8\}), \\
& \qquad S_6(TT = C, CT = \{S_9\})\}, \\
& DT = \text{Alternative}, \\
& TO = S_4 \diamond S_5 \diamond S_6)\}, \\
DT = & \text{No-dependency}, \\
TO = & G_{\text{airlines}} \square S_3 \square G_{\text{hotel}}).
\end{aligned}
$$

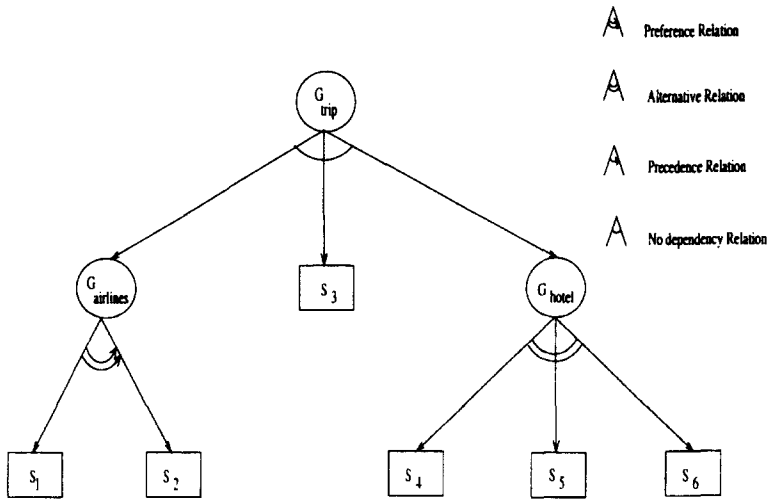

Fig. 1. A transaction tree representation of Example 1.

3. AN EXECUTION ARCHITECTURE FOR THE PROPOSED TRANSACTION MODEL

Figure 2 illustrates the architecture of the transaction execution model for our MDBS. Local transactions are directly submitted to the LDBSs, while global transactions use a common MDBS interface. A global transaction, submitted to the global transaction manager (GTM), is divided into a number of subtransactions, and each subtransaction is sent to the relevant site where the required data pages reside. A set of application programs called *agents* is built on top of the LDBSs to act as an interface between GTM and each local site in controlling the execution of subtransactions.

The objectives of GTM are to avoid inconsistent retrieval of data, and to preserve global consistency and atomicity. These objectives are difficult to achieve because [9]:

- LDBSs are not aware of each other and the MDBS,
- both local transactions and subtransactions can run concurrently at each site,
- LDBSs do not export any concurrency control information to GTM,
- from the LDBSs' point of view, a subtransaction is not different from a local transaction.

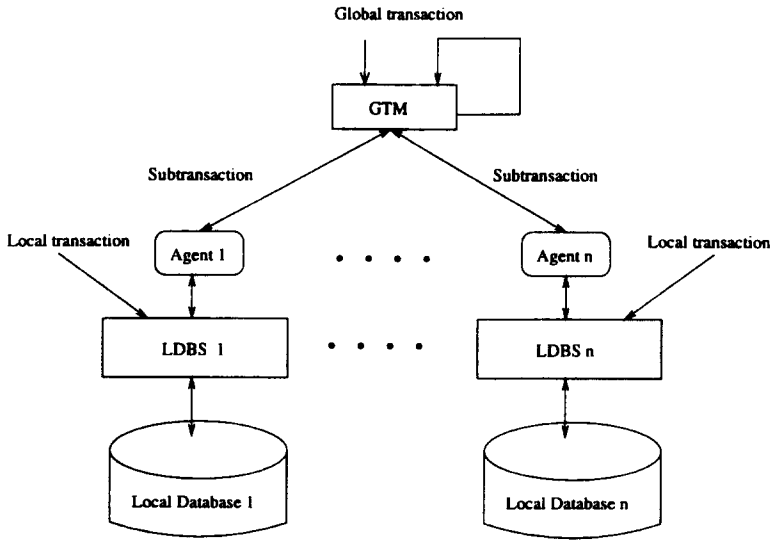

Fig. 2. The transaction execution architecture of the MDBS.

LDBS at each site ensures the local consistency and isolation properties by generating serializable schedules. Global serializability can be provided by obtaining the information of relative serialization order of subtransactions at each local site, and guaranteeing the same relative order at all of those sites [15]. Achievement of global serializability is difficult in an MDBS due to the reasons listed above.

The ticketing method proposed in [9] provides that the serialization order of subtransactions at a local site can be determined at the global level without violating the autonomy of that site. The ticketing method uses a regular data object, called a ticket, to determine the serialization order of subtransactions. A ticket in a database can be considered as a logical timestamp. One ticket value is maintained at each local site. GTM forces each subtransaction to read, increment, and update the ticket value at the site it executes. Ticket values obtained at a site reflect the relative serialization order of subtransactions at that site.

Accomplishing the atomicity of global transactions is another problem in MDBS transaction management. In traditional distributed database systems, atomicity can be achieved by using the two-phase commit (2PC) protocol. In an MDBS, due to the heterogeneity of local components, we cannot expect every participant site to support 2PC. One possible solution to this problem is to use a simulated 2PC protocol. The techniques that can be used to achieve the simulated 2PC are described in [9].

In our transaction execution model, we assume that each global transaction has at most one subtransaction at each local site. We also assume that a local transaction or a subtransaction consists of four basic operations: $r(x)$, $w(x)$, c, and a. $r(x)$ and $w(x)$ are read and write operations on data page x, and c and a are commit and abort operations. A transaction is assumed to be in the ready-to-commit state after it completes all of its read and write operations. It stays in this state until a commit or an abort operation is issued.

In the following sections, we discuss how global atomicity and global serializability are achieved in our execution model.

3.1. ENSURING GLOBAL ATOMICITY

In an MDBS environment, a relaxed version of atomicity, namely the *semantic atomicity*, is discussed in [12] and [19]. In traditional distributed database systems, a global transaction can be atomic if either all or none of its subtransactions complete their execution successfully. A global transaction can commit if all of its subtransactions commit; otherwise, the effects of committed subtransactions are undone, and global transaction is

aborted. In semantic atomicity, on the other hand, subtransactions can commit at different times [8]. We need to extend the traditional atomicity to capture the semantics of dependency relations among subtransactions. The execution of a global transaction G preserves the semantic atomicity if the following conditions are satisfied:

- When a precedence or a no-dependency relation exists among its children, G can commit if all of its child transactions have committed. If one of its child transactions is aborted, G is aborted, and the other child transactions are either aborted or the effects of committed ones are undone.
- If an alternative or a preference relation exists, G can commit if one of its child transactions commits.[3] When a child transaction commits, other child transactions that are executing are aborted.

The execution of a global transaction containing only ordinary children[4] proceeds as follows.

- GTM spawns the children of the global transaction according to the specified dependency type:
  - — If either a no-dependency, or an alternative, or a preference dependency exists, all of the child transactions are created.
  - — Otherwise (if a precedence relation is specified), the children are created on the basis of the given total order.
- If GTM reaches a leaf node in the nested transaction tree and creates a subtransaction, it submits the subtransaction to the corresponding site through the agents.
- When a subtransaction finishes its database operations, the agent of that site sends a ready-to-commit message to GTM.
- After receiving a ready-to-commit message for a subtransaction, GTM checks the dependency type associated with the parent of the subtransaction to find out what to do next.
  - — If a precedence relation exists among its children, the next child transaction in the given order is created by GTM. If all of the child transactions enter the ready-to-commit state, the parent also enters the ready-to-commit state.

---

[3] Remember that, with the preference relation, if $S_i \triangleright S_j$, S_j can be committed only if S_i fails.

[4] The execution of a global transaction that can have commit-independent (compensatable/retriable) transactions is described in Section 3.3.

  - If an alternative relation exists, the parent enters the ready-to-commit state, and GTM sends messages to the relevant agents to abort the other child transactions.
  - If a preference relation exists, the parent enters the ready-to-commit state if the completed subtransaction is the most preferred one. When the parent becomes ready to commit, GTM broadcasts the abort message for the other child transactions.
  - If a no-dependency relation exists, the execution state of the parent becomes ready-to-commit after all of its children enter the ready-to-commit state.
- If the root transaction enters the ready-to-commit state, GTM decides to commit or abort the transaction according to the concurrency control algorithm executed.
- After a commit or abort is issued for the root transaction, GTM broadcasts a message to child transactions down to the leaves of the transaction tree to commit or abort the subtransactions at local sites.

3.2. ENSURING GLOBAL SERIALIZABILITY

The global serializability is ensured in our execution model by employing ticketing-based concurrency control for global transactions. The ticket values obtained by subtransactions are transferred to their parents up to the root transaction. GTM ensures the same relative serialization order at all sites of the global root transaction using the ticket values obtained. Two possible methods that can be used to control concurrent execution of global transactions are the optimistic ticketing method and the conservative ticketing method [9]. The following two subsections will describe the implementation details of these two methods in our execution model considering only ordinary subtransactions. The execution strategies for commit-independent subtransactions will be detailed in Section 3.3.

3.2.1. Employing the Optimistic Ticketing Method (OTM)

OTM allows subtransactions of global transactions to be executed as soon as they are submitted to the local sites. A global transaction is committed when all of the ticket values obtained by its subtransactions have the same relative order at all participant LDBSs.

If OTM is adopted to our execution model, a global transaction G is processed as follows:

- First, a time-out period[5] is set for G.
- The GTM spawns the child transactions of G, according to the rules given in the preceding section, up to the subtransactions executed at local sites.
- Subtransactions are allowed to execute under the control of agents until they become ready-to-commit.
- When G enters the ready-to-commit state, it is validated by GTM.
- If the validation of G is successful, it is committed; otherwise, it is aborted and then restarted.
- If the time-out of G expires before its validation test starts, it is aborted and then restarted.

GTM uses a global serialization graph (GSG) to validate the commitment of transaction G. GSG is a directed graph whose nodes correspond to the recently committed global transactions. For any pair of recently committed global transactions G_i and G_j, there is a directed edge $G_i \rightarrow G_j$ if G_i has obtained a smaller ticket value than G_j at a site where they were executed together.

A global transaction G in the ready-to-commit state is validated as follows.

- First, a node is created for G in GSG.
- If G has obtained a smaller (larger) ticket value than a recently committed global transaction G_c at a site, an edge $G \rightarrow G_c$ ($G_c \rightarrow G$) is inserted.
- If all such edges can be added to GSG without creating a cycle, G is validated.
- Otherwise, the node for G and all related edges are removed from the graph, and G is aborted.

A validation can be performed on GSG either:

- when a global child transaction becomes ready-to-commit (i.e., *early validation*), or
- when a global root transaction becomes ready-to-commit (i.e., *late validation*).

[5] For the detection of a potential deadlock.

The aim of early validation is to detect the conflicts among global transactions as early as possible and to minimize the global transaction restarts. If a global child transaction fails in GSG test, GTM can abort that transaction. If a preference or an alternative dependency relation exists among the transactions that belong to the same parent, GTM can execute an alternative transaction for the failed child transaction. If a no-dependency relation or a precedence relation exists, GTM restarts the aborted global child transaction.

If the violation test for a global root transaction is successful, a commit message is transmitted to its children. Otherwise, an abort message is sent to its children, and the entire global transaction is restarted.

To remove a node for a committed global transaction G from GSG, the following properties should be satisfied [9]:

- The node has no incoming edges.
- The transactions that were active when G was committed have all been terminated.

3.2.2. Employing the Conservative Ticketing Method (CTM)

CTM was introduced to eliminate the global restarts experienced by OTM due to the ticketing conflicts. CTM controls the order in which the subtransactions take their ticket values. In order to apply CTM, we need an additional ready-to-take-a-ticket state of transactions. A subtransaction enters the ready-to-take-a-ticket state after it completes all of the database operations before obtaining its ticket value. The agents over the local sites are responsible to detect ready-to-take-a-ticket states of subtransactions and send related messages to GTM.

If CTM is employed in our system, a global transaction is processed as follows:

- Initially, a time-out period is set for each global transaction.
- A subtransaction is allowed to execute under the control of LDBSs until it enters the ready-to-take-a-ticket state.
- When a subtransaction enters the ready-to-take-a-ticket state, the agent of that site sends a ready-to-take-a-ticket message to GTM.
- After receiving this message for a subtransaction, GTM checks the dependency type associated with the global parent of the subtransaction to determine whether the parent should also enter the ready-to-take-a-ticket state. This determination is based on the execution rules specified for the ready-to-commit message in Section 3.1.

- Global transactions, which are determined to enter the ready-to-take-a-ticket state, are allowed to take their ticket values according to the order in which they enter this state. If a global transaction G_1 becomes ready to take a ticket before another transaction G_2, G_1 is assigned a smaller ticket value than that of G_2.
- A global transaction that enters the ready-to-commit state is committed by GTM. If the time-out of a global transaction expires before it is committed, the transaction is aborted and then restarted.

3.3. COMMIT-INDEPENDENT SUBTRANSACTIONS

The preceding section has provided the implementation details of the execution model for ordinary subtransactions. In this section, we consider commit-independent (i.e., compensatable and retriable) subtransactions. The execution strategies detailed in both sections can be used together for a mixture of ordinary and commit-independent subtransactions. For clarity of presentation, we preferred to discuss the two types of subtransactions in two separate sections. Before the description of the execution model for commit-independent subtransactions, let us first specify the necessary assumptions and restrictions for the underlying MDBS environment:

- There should be no value dependencies among the commit-independent subtransactions.
- If a compensating transaction is initiated, it completes successfully.

The aim of commit independent subtransactions is to reduce the blocking effect of the 2PC global atomic commitment protocol. If a subtransaction commits before its parent, it is called an *early committed subtransaction*. Similarly, if a subtransaction commits after its parent, it is a *late committed subtransaction*. Compensatable subtransactions can be early committed, and retriable subtransactions can be late committed. To achieve semantic atomicity with commit-independent subtransactions, the following conditions should hold for a global transaction G [12]:

- If G is aborted, the effects of early committed subtransactions of G on the database are not seen by other transactions.
- If G is committed, the effects of its late committed subtransactions are seen by the transactions serialized after G.

Consequently, for a compensatable subtransaction S with its compensating transaction CS, if the parent of S is aborted, commitment of S is required to be undone by executing CS. The effects of committed subtransactions are not seen if no other subtransaction is serialized between

the S and CS. Therefore, if GTM ensures that no other subtransaction takes its ticket value before the commitment of CS, consistency of the MDBS is preserved.

The compensating transaction execution is handled by agents. If a global transaction G has a compensatable subtransaction S with its associated compensating transaction CS, the execution of S proceeds as follows when OTM is employed:

- CS is sent to the relevant agent with the submission of S.
- When S enters the ready-to-commit state,
  - — The agent sends a ready-to-commit message to GTM.
  - — The ticket value obtained by S is recorded, and S is early committed by the agent.
- If the agent receives an abort message for S after it has been early committed, it submits CS to LDBS. The agent sends an abort message for the subtransactions that has obtained a ticket value greater than that of S before CS is committed.

If CTM is being used for global concurrency control:

- CS is sent to the relevant agent with the submission of S.
- When S enters a ready-to-take-a-ticket state, the agent sends a ready-to-take-a-ticket message to GTM.
- The agent does not permit other subtransactions to enter their ready-to-take-a-ticket states until S takes its ticket.
- If S successfully takes its ticket and completes all of its operations, the agent early commits S and sends a ready-to-commit message to GTM.
- The agent does not allow other subtransactions to take their ticket values until S is committed or aborted.
- If an abort is issued for the early-committed S, the agent submits CS to the LDBSs, and does not submit any other subtransaction operation until CS is committed.

In the case of retriable subtransactions, the global transactions do not see an inconsistent database if GTM avoids serialization of any subtransaction between the commitment of a global transaction and the commitment of a retriable subtransaction that belongs to the committed global transaction. A global transaction G that contains a retriable subtransaction RS can be committed without waiting RS to finish its execution. GTM can commit G, while RS is still being executed at a site, but it does not permit another subtransaction to take a ticket at that site until RS takes its ticket.

If OTM protocol is employed, the execution of retriable subtransaction *RS* of *G* can be handled as follows:

- If *G* enters the ready-to-commit state before a ready-to-commit message arrives for *RS*, a $+\infty$ ticket value is used for *G* in GSG test. Since the RS has not taken its ticket yet, the $+\infty$ ticket value in GSG test ensures that no other subtransaction is serialized between the commitment of *G* and the commitment of *RS*.
- When *RS* is committed, the agent sends a commit message to GTM in order to update the ticket value of *G*.

If the employed protocol is CTM, agents simply do not allow other subtransactions to take their tickets until *RS* is successfully committed. The execution protocol for a retriable subtransaction *RS* can be described as follows:

- Once the agent receives a take-a-ticket message for *RS*, it does not send ready-to-take-a-ticket messages for other subtransactions executed at that site until *RS* takes its ticket successfully.
- GTM makes the state of RS ready-to-commit after sending a take-a-ticket message to it.
- If GTM issues a commit for *RS*, the agent does not submit the ticket operation of other subtransactions to the LDBS until *RS* is committed.

4. SIMULATION MODEL

With the simulation experiments of this section, we aimed to investigate the performance implications of MDBS transaction management. Various experiments have been conducted to evaluate the cost of transaction processing in an MDBS environment. The performance of OTM and CTM algorithms built on the proposed transaction model has also been evaluated.

Reliability and recovery issues are not considered in our simulation model. We assume a reliable system, in which no site failures or communication network failures occur. The other assumptions of the simulation model can be listed as follows:

- LDBSs can abort a transaction that executes at its site to recover from a local deadlock.

- LDBSs notify the transaction programs when unilaterally abort a transaction. This means that MDBS will be aware of subtransaction aborts at local sites.
- Subtransactions have a visible ready-to-commit state.

The architecture of our MDBS that was adapted from [10] is illustrated in Figure 3. An MDBS is modeled as a closed network with a single global site and a number of local sites. A global transaction manager (GTM) resides at the global site, and it acts as a server to global clients. Global transactions are submitted to the system through the GTM interface. There exists one LDBS residing at each local site. A global transaction agent (GTA) is built on top of each LDBS. GTM sends the global subtransactions to GTAs of the relevant sites. Each GTA is responsible for submitting global subtransactions to its LDBS, as well as communicating with GTM. Local clients submit local transactions to the LDBS at their sites.

The configuration parameters of our MDBS model are listed in Table 1. It is assumed that each global or local client submits its transactions one after another. The size of the local database is assumed to be the same at each site. A page is considered as the unit of data access. Each data page in the database is simulated individually. The simulation program also keeps track of the list of data pages resident in the main memory of each site.

Each local transaction processed at a site contains read and write operations on data pages stored at that site. The parameters associated

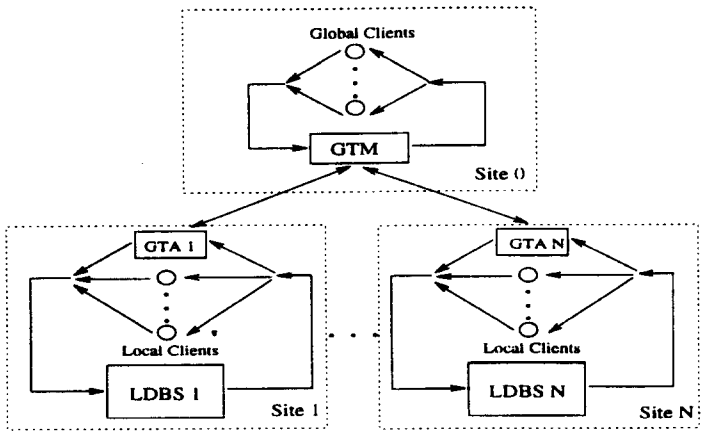

Fig. 3. MDBS architecture.

TABLE 1
Configuration Parameters

NumSites	Number of local sites
GNumClient	Number of global clients
LNumClient	Number of local clients at each site
LDBsize	Size of each local database (in pages)
LMemsize	Main memory size at each site (in pages)

with local transactions are described in Table 2. The local transaction size (i.e., *LTranSize*) refers to the number of data page accesses of a local transaction. Each data page accessed is updated with a probability of *LWriteProb*. When a local transaction execution is completed, a new local transaction is generated at that site after a think time which is specified by parameter *LThinkTime*.

The global transaction parameters used in our simulation model are listed in Table 3. As discussed in the previous sections, a global transaction can be modeled as a tree where the internal nodes are global transactions and the leaf nodes are subtransactions. The maximum height of a global transaction tree and the maximum number of child transactions at each internal node are specified by parameters *TreeHeight* and *NumChild*, respectively. The maximum number of subtransactions executed in a global transaction is then $(NumChild)^{TreeHeight}$. Since we assume that at most one subtransaction of a global transaction can be executed at each site, the number of subtransactions of a global transaction also determines the number of local database sites that the global transaction may access.

The dependencies among the children of a global transaction are determined by the probabilities of different dependency types. To analyze the effects of compensating and retriable transactions, *OrdinaryProb*, *CompensatableProb*, and *RetriableProb* parameters are defined. These parameters determine, on the average, the ratio of subtransactions' types in the overall global transaction. The values of the three parameters at any instant should sum up to 1. Similar to a local transaction, each subtransac-

TABLE 2
Local Transaction Parameters

LTranLen	Local transaction length in pages
LWriteProb	Data update probability at a local transaction access
LThinkTime	Think time of a local client

TABLE 3
Global Transaction Parameters

TreeHeight	Maximum height of a global transaction tree
NumChild	Maximum number of children in each global transaction
AlternativeProb	Probability of alternative relation
PreferenceProb	Probability of preference relation
PrecedenceProb	Probability of precedence relation
OrdinaryProb	Probability of ordinary subtransactions
RetriableProb	Probability of retriable subtransactions
CompensatableProb	Probability of compensatable subtransactions
GTranLen	Subtransaction length in pages
GWriteProb	Data update probability at a subtransaction access
GThinkTime	Think time of a global client

tion contains a number of read and write operations on data pages. Ticketing operations of a subtransaction are assumed to be read and write operations on a specific page.

Resource-related parameters used in the simulation model are described in Table 4. The delay of communication messages and the processing cost of those messages (at both the source and destination sites) are explicitly simulated by using the parameters *MessTransTime* and *CPUMessTime*, respectively. A resource unit at each local site is modeled by one CPU and two disks. Each site is assumed to have an equal number of resource units, which is determined by the parameter *LResourceUnit*.

4.1. SIMULATION MODEL COMPONENTS

In this subsection, we describe the simulation model components in more detail. An illustration of the simulation model is provided in Figure 4. Each component of the model can be described as follows:

- Global Transaction Generator (GTG): GTG simulates the global client behavior by generating global transactions on the basis of the

TABLE 4
Resource Parameters

MessTransTime	Delay of a communication message
CPUMessTime	CPU time to process a communication message
LResourceUnit	Number of resource units at each site
LCPUTime	CPU time to process one data page
LDiskTime	Disk time to read/write one data page

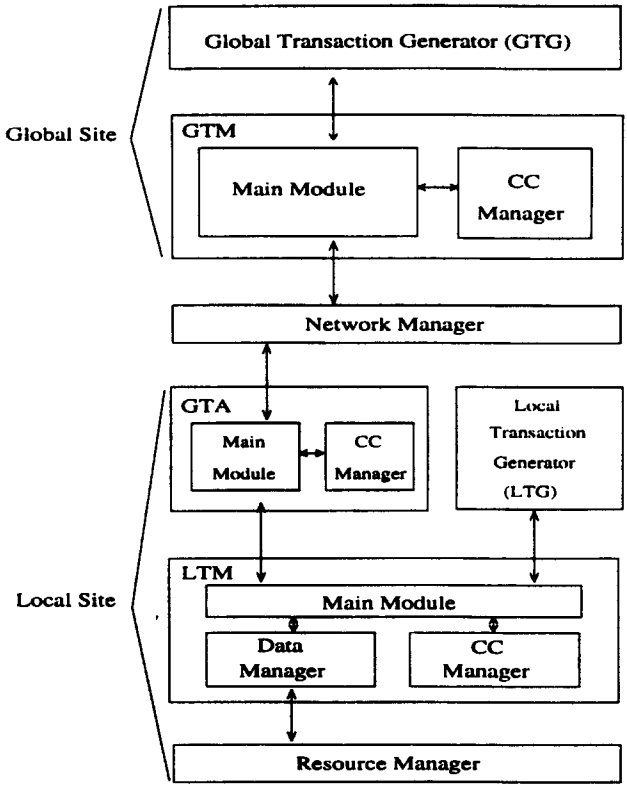

Fig. 4. Simulation model components.

parameters listed in Table 3. At the beginning of the simulation, *GNumClient* global transactions are created and submitted to GTM. During the simulation, a new transaction can only be created after the termination of a global transaction.

- Global Transaction Manager (GTM): GTM accepts global transactions from GTG and models their execution. It consists of two modules:

  — Main Module: This module models the transaction execution with the help of the Concurrency Control (CC) manager. There are two main functions of this module. First, it accepts global transactions and decomposes them into their subtransactions executed at each local site according to the rules discussed in Section 3. Second, it establishes a 2PC protocol with agents and coordinates incoming and outgoing messages for subtransactions. When a

global transaction enters the ready-to-commit state, it decides either commitment or abortion of that transaction after communicating with the CC manager.

— Concurrency Control (CC) Manager: The CC manager models the execution of a concurrency control algorithm for serialization of global transactions. It also performs deadlock detection if necessary. This module enables us to plug in different global concurrency control and deadlock detection algorithms for performance studies.

- Local Transaction Generator (LTG): This component simulates the local client behavior and generates local transactions based on the values of parameters provided in Table 2. Similar to GTG, it submits a new transaction when one of the previously submitted local transactions completes its execution.
- Global Transaction Agent (GTA): The GTA resides at each local site and models the execution of global subtransactions at that site. Similarly to GTM, it consists of two modules.

  — Main Module: The GTA main module is responsible for controlling the submission of subtransactions at its site. It determines the submission time of a subtransaction's operations with the help of the CC manager and the messages coming from GTM. Submission time of the ticketing operation is also determined by the GTA main module based on the local concurrency control algorithm. If the local concurrency control algorithm is based on locking, GTA submits the ticketing operations at the end of each transaction execution [9]. The main module also handles the submission of compensating subtransactions by interacting with the CC manager.
  — CC Manager: It is the local agent part of the concurrency control algorithm implemented in GTM. It carries out the global concurrency control for subtransactions at its site.

- Network Manager: It models the network resource connecting the local sites and the global site.
- Local Transaction Manager (LTM): The LTM accepts and models the execution of local transactions and subtransactions. It consists of three modules.

  — Main Module: The main module models the local transaction execution by interacting with the CC manager and data manager.
  — CC Manager: The CC manager models the local concurrency control algorithm as well as the local deadlock detection algorithm.

— Data Manager: The data manager models data accesses by interacting with the resource manager and the main module of LTM.

- Resource Manager: This component models CPU and disk accesses at its site.

5. SIMULATION EXPERIMENTS

Our performance model was implemented on a simulation testbed using the CSIM simulation package from MCC [17]. Each run of the simulation experiments was continued until 5000 transactions (global + local) were processed in the system. The *independent replication* method was used to validate the results by running each configuration ten times with different random number seeds and using the averages of the replica means as final estimates. 90% confidence intervals were obtained for the performance results. The width of the confidence interval of each statistical data point is less than 4% of the point estimate. In displayed graphs, only the mean values of the performance results are plotted.

In the experiments, we employed the basic two-phase-locking (2PL) concurrency control algorithm [2] at the local sites. For the detection of local deadlocks, a local wait-for graph was maintained at each site. To handle global deadlocks, we implemented a time-out mechanism for the execution of global transactions.

The basic performance metrics used are *global* (*local*) *throughput* (i.e., the number of committed global (local) transactions per second), and *global* (*local*) *abort ratio* (i.e., the number of global (local) transaction aborts over the total number of global (local) transactions submitted to the system). The throughput results are also indicative of how the response time trends would be for the transactions.

Simulation experiments were driven by the parameter values presented in Table 5. The parameter values were chosen so as to be comparable to the related simulation studies such as [1, 10]. The workload model used in the experiments simulates an environment in which there exist some amount of data and resource contention among the global and local transactions. All of the local sites are assumed identical and operating under the same parameter values.

We observed in our preliminary experiments that using an adaptive restart delay of an average global (local) transaction response time for the aborted global (local) transactions gives the best results.

In the experiments of the following sections, we investigate the performance impact of OTM and CTM algorithms and the extended transaction model characteristics.

TABLE 5
Default Values for Configuration and Workload Parameters

NumSites	8 sites
GNumClient	20 clients
LNumClient	30 clients
LDBSize	1000 pages
LMemsize	250 pages
LTranLen	8 pages
LWriteProb	0.25
LThinkTime	0 ms
TreeHeight	1, 3
NumChild	2
AlternativeProb	0.00
PreferenceProb	0.00
PrecedenceProb	0.00
OrdinaryProb	1.00
RetriableProb	0.00
CompensatableProb	0.00
GTranLen	8 pages
GWriteProb	0.25
GThinkTime	0 ms
MessTransTime	5 ms
CPUMessTime	20 ms
LResourceUnit	5
LCPUTime	100 ms
LDiskTime	200 ms

5.1. EVALUATION OF OTM AND CTM CONCURRENCY CONTROL ALGORITHMS

The comparative performance of OTM and CTM global concurrency control algorithms was evaluated under different levels of data contention by varying parameter *GWriteProb*. The global throughput results obtained with the algorithms are displayed in Figure 5. Under very low levels of data contention (i.e., when the workload consists only of read-only transactions), OTM performs better than CTM. The worse performance of CTM can be contributed to the overhead of blocking a completed subtransaction until all of its siblings become ready to take a ticket. However, as the data contention among transactions increases, the performance of OTM becomes worse than that of CTM. This result is due to the increasing number of validation aborts with OTM. Figure 6 presents the abort ratios of both algorithms under different levels of data contention. OTM has a higher abort ratio due to validation aborts. The other types of aborts, i.e., deadlock recovery and time-out aborts, are experienced with both OTM and CTM.

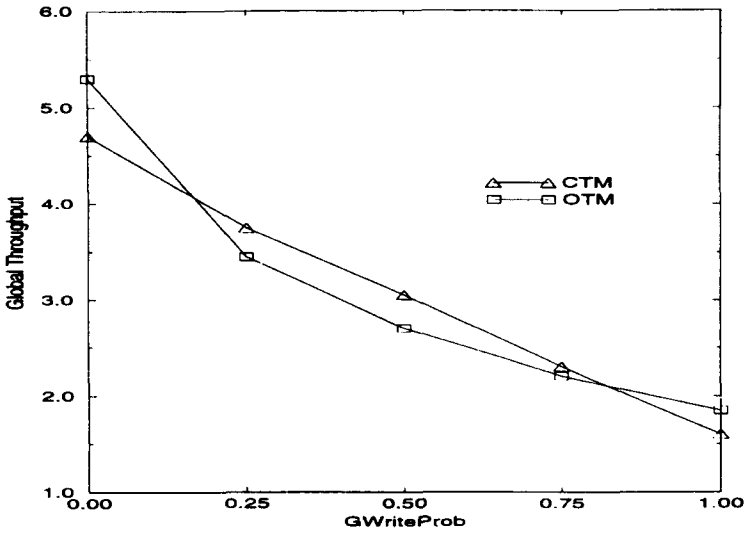

Fig. 5. Global throughput versus global write probability.

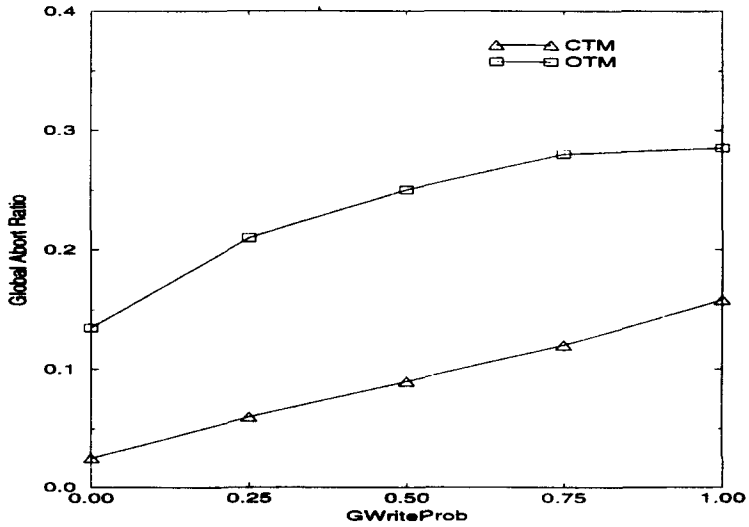

Fig. 6. Global abort ratio versus global write probability.

Under high levels of data contention, OTM again outperforms CTM. This result can be explained by the fact that the increasing blocking times of subtransactions (due to higher data contention) lead to a large number of global and local deadlocks with algorithm CTM.

We also evaluated the algorithms under different levels of resource contention by varying the parameter *LResourceUnit*. The performance of the algorithms improved when the number of hardware resources was increased; however, the comparative performance of the algorithms was similar to what we observed under various levels of data contention.

5.2. *IMPACT OF DEPENDENCY RELATIONS*

In this set of experiments, we investigated the performance impact of each dependency relation type individually. Figure 7 illustrates the effects of processing various amounts of alternative relation transactions[6] on the performance of global transactions with both OTM and CTM algorithms. It is assumed that there exist no other types of dependencies among

[6] Here, we again use the term "transaction" to mean either subtransaction or global transaction.

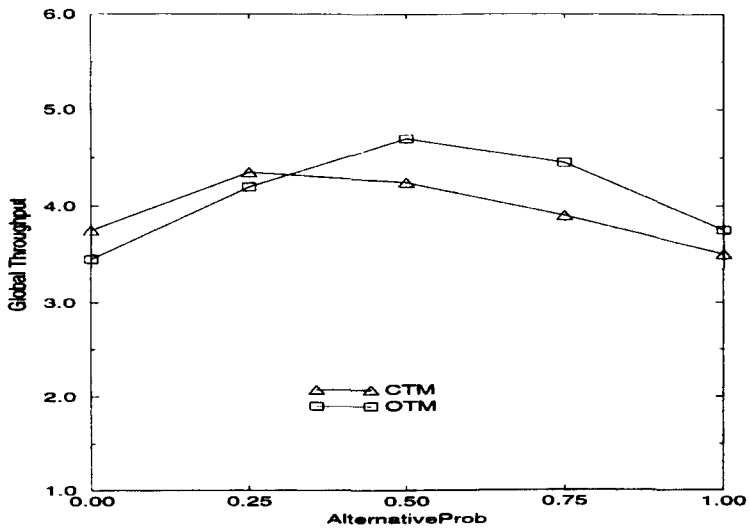

Fig. 7. Global throughput versus probability of alternative dependency relation.

transactions (i.e., *PreferenceProb* and *PrecedenceProb* parameters are set to 0). As the number of transactions with alternative dependency increases, the global throughput of the system also increases up to a certain point; however, increasing the number of such transactions beyond that point results in worse performance. This means that having the alternative dependency for some of the transactions can improve the performance since the overhead of restarting transactions is avoided. However, if the system runs alternative child transactions for most of the submitted global transactions, then the performance benefit of avoiding restarts is outweighed by the increased data and resource contention among the large number of transactions processed concurrently in the system.

For small numbers of alternative transactions, the CTM algorithm outperforms OTM. Remember that, in the preceding section, we obtained the same result with *GWriteProb* = 0.25 (i.e., the default value used in the experiments of this section), and explained this result by the validation aborts experienced with OTM. For the *AlternativeProb* values that are greater than 0.25, the performance of OTM becomes better than that of CTM. For those values, the amount of transaction aborts avoided by executing alternative transactions becomes large enough to affect the comparative performance of OTM and CTM. The performance impact of the primary drawback of OTM, i.e., the overhead of validation aborts, is reduced by the alternative transactions. Figure 8 displays the abort ratios

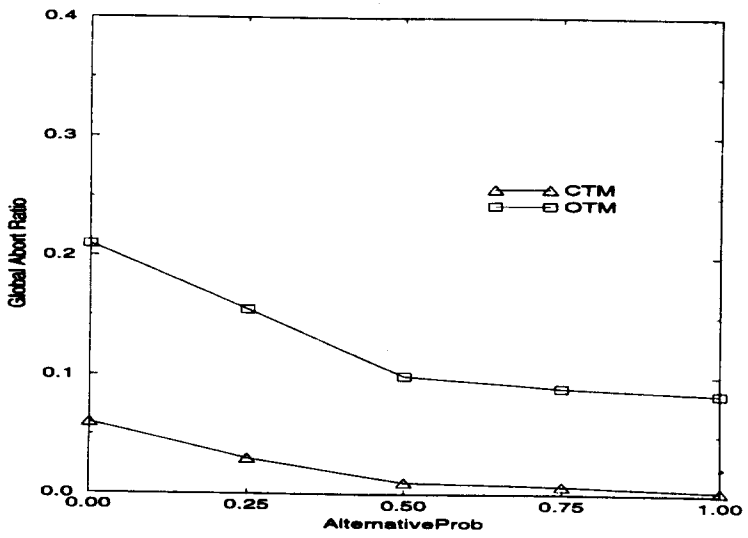

Fig. 8. Global abort ratio versus probability of alternative dependency relation.

obtained with both algorithms as a function of the fraction of alternative transactions. Although the number of transaction aborts with OTM is still larger than that of CTM, it is reduced to a great extent compared to the results obtained with no alternative transactions (Figure 6). The worse performance of CTM is due to the blocking times of completed subtransactions before getting their tickets, and the higher chances of transaction deadlocks.

The throughput results of the experiment that evaluated the effects of preference dependency among transactions is provided in Figure 9. The comparative performance trends of OTM and CTM algorithms is similar to that we obtained with the alternative dependency. This is an expected result since, as explained in previous sections, the preference relation is implemented by executing concurrently all of the transactions that are alternative to the preferred transaction. Therefore, the discussion we have provided above for the relative performance of OTM and CTM with the alternative relation is also valid for the preference relation. However, as a difference from the results obtained with the alternative dependency, a very slight throughput improvement is observed by increasing the fraction of transactions with preference dependency up to a value of 0.25. Also, the throughput obtained in general with the preference relation is at a lower

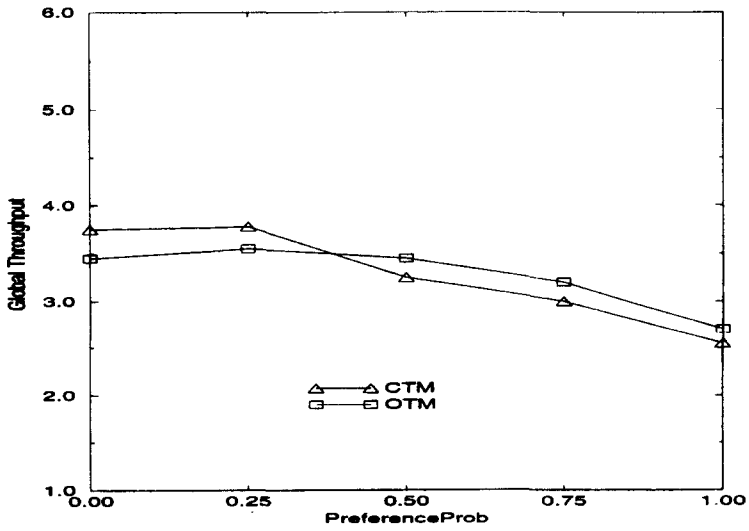

Fig. 9. Global throughput versus probability of preference dependency relation.

level compared to that with the alternative relation. All of these results can be contributed to the higher overhead of executing transactions with the preference dependency. In implementing the preference dependency, GTM submits all of the alternative transactions; however, it waits for the completion of the preferred one. Transactions that are alternative to the preferred transaction and have completed their execution are not committed unless the preferred one is aborted due to some reason. This will result in increased response times and lower throughput for global transactions.

When we increased the number of resource units at each site, we observed better performance by issuing transactions with alternative and preference dependencies. This result is obvious because the overhead of processing a large number of extra transactions is reduced by providing them with surplus resources.

Figure 10 presents the throughput results obtained with different degrees of precedence dependency. Increasing the number of transactions with precedence dependency results in a performance degradation with both algorithms OTM and CTM. The precedence dependency among transactions leads to an increase in the response times, as dependent transactions have to wait for the commitment of some other transactions. Therefore, each additional precedence dependency affects the throughput negatively. As another difference from the results of other types of dependencies, the performance of OTM never becomes better than the

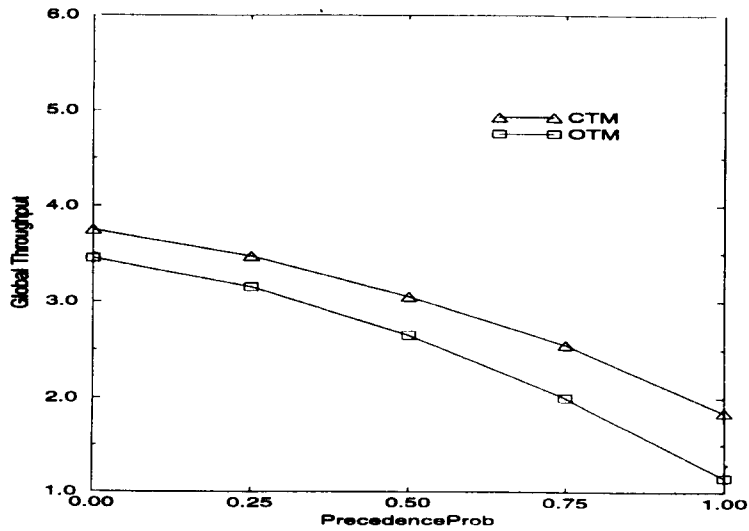

Fig. 10. Global throughput versus probability of precedence dependency relation.

performance of CTM when there exists precedence dependency among some transactions. Since this type of dependency leads to an increase in the response times of transactions, the impact of validation aborts experienced with OTM is more crucial. Although the blocking times and the probability of deadlocks also increase with more precedence dependency, and this affects the performance of CTM negatively, the overhead of the large number of validation aborts becomes the determining factor for the relative performance of OTM and CTM.

5.3. *IMPACT OF COMMIT INDEPENDENT SUBTRANSACTIONS*

In the experiments of this section, we investigated how compensatable and retriable subtransactions affect the performance of global transactions. We set parameters *AlternativeProb*, *PreferenceProb*, and *PrecedenceProb* to 0 to isolate the effects of transaction dependencies. In the first experiment, we evaluated the performance impact of retriable transactions by varying parameter *RetriableProb* from 0.00 to 1.00 in steps of 0.25. All of the subtransactions that are not retriable are assumed to be of type ordinary.

Figure 11 illustrates the throughput results of global transactions for different ratios of retriable subtransactions with algorithms OTM and

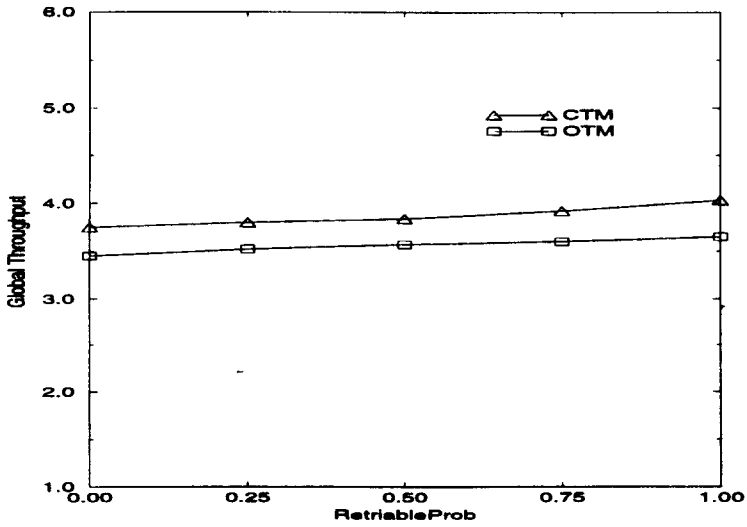

Fig. 11. Global throughput versus probability of retriable subtransactions.

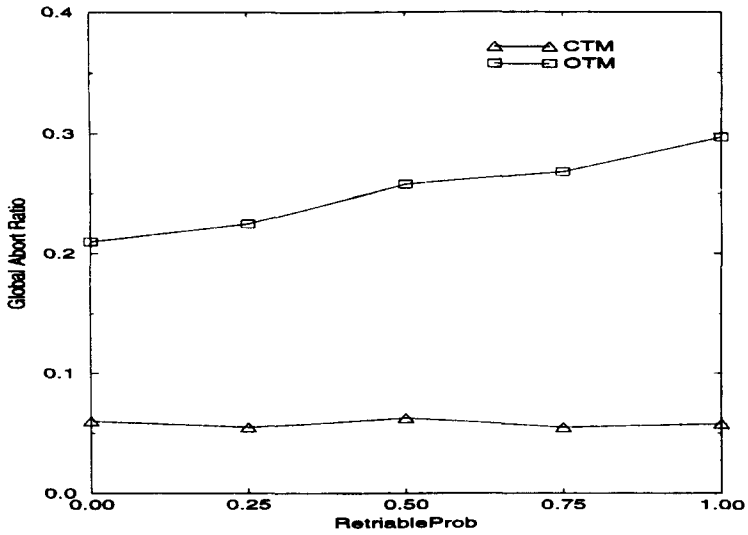

Fig. 12. Global abort ratio versus probability of retriable subtransactions.

CTM. A slight improvement is observed in the performance of both algorithms as the probability of retriable subtransactions increases. Although it is expected that retriable subtransactions can provide substantial improvements in the performance since they provide global transactions to commit early, this expectation is not confirmed by the results. With OTM, the performance advantage gained by early global commits is outweighed by the increased number of transaction aborts during validation tests. It should be ensured by this algorithm that no other transaction is serialized between a global transaction and its retriable subtransaction. Failure to satisfy this condition can be the source of many validation aborts. Abort ratios of the algorithms are displayed in Figure 12. Although the abort ratios of transactions are not affected by retriable subtransactions with CTM, the throughput improvement is not substantial with this algorithm either. The overhead in this case is the blocking times experienced with transactions. Although a global transaction with a retriable subtransaction is allowed to commit early, the CTM algorithm requires that the other subtransactions running concurrently at the same site wait for that subtransaction until it takes its ticket. The comparative performance of OTM and CTM does not seem to be affected by the amount of retriable subtransactions.

The impact of compensatable subtransactions on the global throughput is illustrated in Figure 13. We should not expect global performance gains

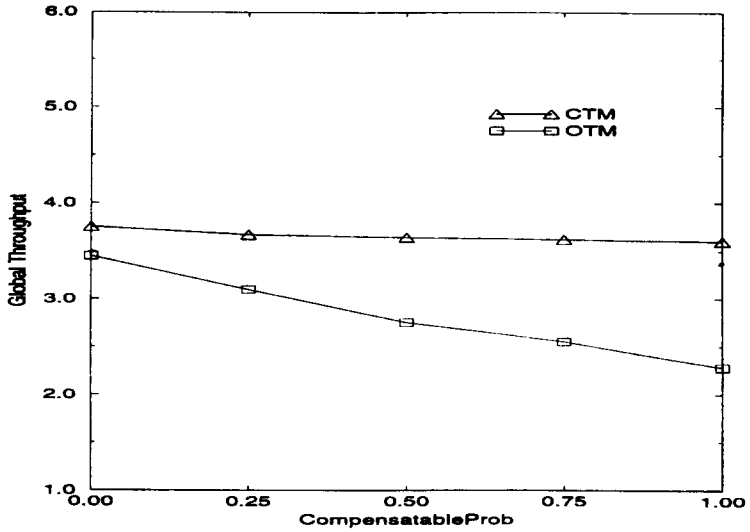

Fig. 13. Global throughput versus probability of compensatable subtransactions.

from the compensatable transactions since the response time of a global transaction does not depend on the execution of its compensatable children.[7] CTM's performance is not much affected by the amount of compensatable subtransactions. The performance of OTM, on the other hand, is affected negatively when the number of compensatable subtransactions is increased. A compensatable subtransaction can be committed earlier than its parent transaction; however, during the validation test of OTM, GTA aborts the subtransactions that have obtained higher ticket values than the compensatable subtransaction before its parents commits. The number of such aborts is large enough to degrade the performance of OTM. The sharp increase in the number of aborts as a function of increasing ratio of compensatable subtransactions can be seen in Figure 14.

5.4. IMPACT OF EXTENDED TRANSACTIONS ON LOCAL TRANSACTIONS

The experiments of this section were conducted to evaluate the impact of various dependency types and global subtransaction types on the perfor-

[7]However, from the local DBMS point of view, early committed subtransactions can improve the local transaction throughput as the locks are released earlier. This prediction is confirmed by the experiments of the next section.

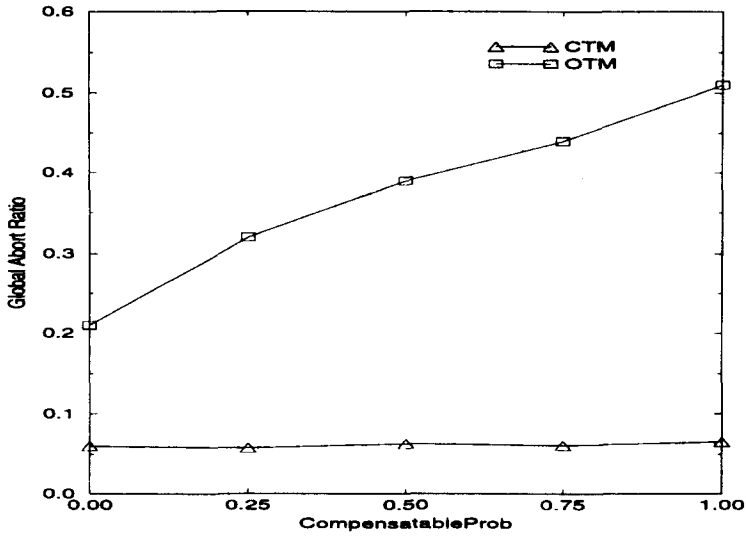

Fig. 14. Global abort ratio versus probability of compensatable subtransactions.

mance of local transactions. It was observed in those experiments that the local transactions perform better in general when OTM rather than CTM is employed to ensure global serializability. With CTM, a subtransaction that has completed all of its operations has to wait for the completion of its sibling subtransactions before getting its ticket. This situation leads to an increase in the number of data conflicts between subtransactions and local transactions. The possibility and the overhead of local deadlocks also increase. Therefore, the performance of local transactions is affected negatively.

In the first set of experiments, we investigated the performance impact of alternative and preference dependency types among transactions. For each of those two dependency types, we observed that an increase in the number of dependent transactions results in a performance degradation for local transactions with both OTM and CTM algorithms. That observation can be explained by the fact that, with both dependency types, the extra subtransactions that are created as alternatives to each other cause an increase in the level of both data and resource contention. Therefore, local transactions experience more data and resource conflicts. Figure 15 presents the local throughput results obtained for the preference relation type. We observed similar performance trends with alternative transactions.

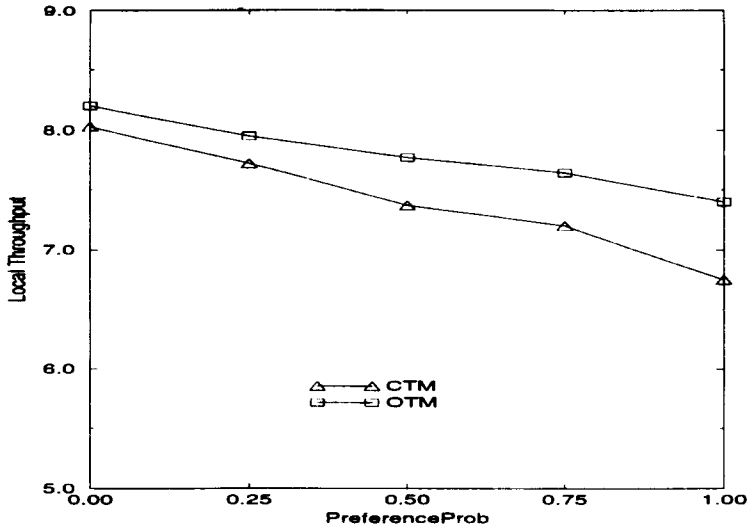

Fig. 15. Local throughput versus probability of preference dependency relation.

Figure 16 shows how the precedence relation affects the performance of local transactions. As the number of transactions with precedence dependency increases, the performance obtained with CTM becomes drastically worse than that with OTM. While processing dependent transactions, OTM allows a completed transaction to release its locks before the dependent transaction starts its execution. With CTM, this is not the case; therefore, local transactions have more data conflicts with subtransactions.

Including different amounts of retriable subtransactions in the system did not have a considerable effect on the performance of local transactions. Increasing the number of retriable subtransactions, and therefore having more global transactions be committed earlier improved the local transaction throughput very slightly. This is because, although some global transactions can commit earlier, their retriable subtransactions continue to contend with local transactions for system resources. The situation, however, was different with compensatable subtransactions. Committing some of the subtransactions earlier and releasing their resources gave a better chance to local transactions to access the required resources without experiencing much contention and to finish early. Figure 17 provides the local throughput results obtained as a function of the fraction of compensatable subtransactions. For a very large number of compensatable subtransactions, OTM cannot provide further improvement in the local performance. This result can be contributed to the GTA aborts that we have discussed at the end of the preceding section.

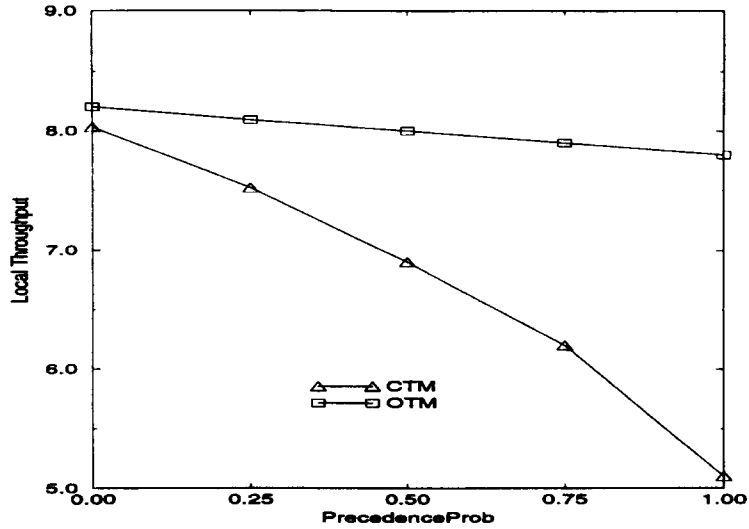

Fig. 16. Local throughput versus probability of precedence dependency relation.

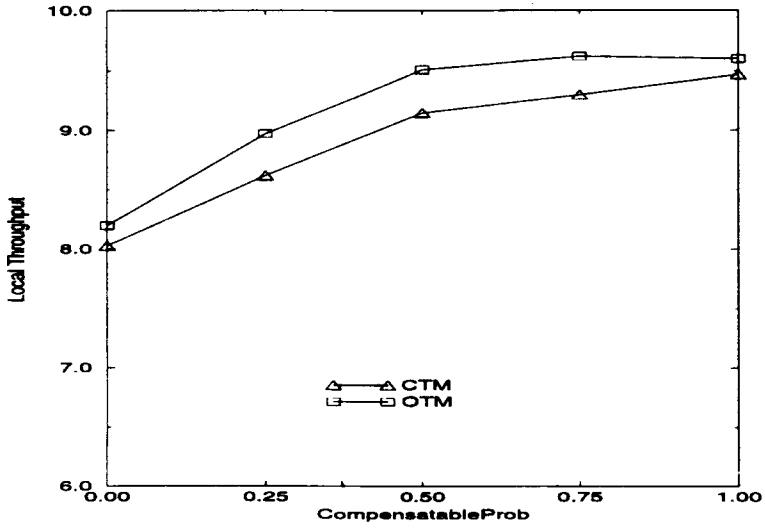

Fig. 17. Local throughput versus probability of compensatable subtransactions.

6. CONCLUSIONS

Most work in the multidatabase systems (MDBSs) area has focused on the issues of transaction management and concurrency control (e.g., [3, 4, 7, 16]). It is difficult to implement traditional transaction management techniques in an MDBS due to the heterogeneity and autonomy of the connected local sites. In this paper, we introduced an extended transaction model for MDBSs. The proposed transaction model covers nested transactions, various dependency types among transactions, and commit-independent subtransactions that make the model much more flexible and powerful than the traditional transaction model. The execution architecture described for transactions does not make any assumption regarding the concurrency control protocols executed at the local sites connected to the MDBS. The global serializability is ensured through the ticketing method proposed by Georgakopoulos *et al.* [9]. Atomic commitment of global transactions is provided through the use of the two-phase commit (2PC) protocol. The blocking effect of 2PC is reduced by executing commit-independent subtransactions.

We provided a detailed simulation model of an MDBS to investigate the performance implications of the proposed transaction model. The simulation model was also used to evaluate the comparative performance of the two variations of the ticketing method: the optimistic ticketing method (OTM) and the conservative ticketing method (CTM). It was observed in the evaluations that the transaction restarts experienced with validation tests constitute a substantial overhead for the response time of transactions when OTM is employed. The primary overhead of CTM, on the other hand, is the blocking delays of transactions prior to the assignment of ticket values. The blocking delays also lead to a large number of transaction deadlocks under high levels of data contention.

When we evaluated the performance impact of transaction dependency types involved in our transaction model, we observed that processing transactions with alternative or preference dependency improves the global throughput of the system if such transactions do not constitute a large fraction of all the transactions processed in the system. Otherwise, the performance benefit provided by avoiding some of the transaction aborts is outweighed by the increased data and resource contention due to processing a large number of extra transactions introduced by the alternative or preference dependency. The situation under high contention was worse with the preference dependency since the additional transactions introduced as alternatives to the preferred transaction do not provide any advantage unless the preferred transaction is aborted due to some reason.

Processing transactions with the precedence dependency degraded the global throughput of the system, as expected, since the level of concurrency is lower with such transactions. Also, the cost of transaction aborts is higher with the precedence dependency, which causes OTM to perform worse than CTM.

The dependency types among global transactions had a negative impact on the performance of local transactions. With the alternative and preference dependencies, the extra transactions created as alternatives to each other lead to an increase in the level of data and resource contention at each local site. Therefore, local transactions experience more data and resource conflicts. The precedence dependency among transactions also causes a degradation in the local transaction throughput. The performance of local transactions with CTM was worse than that with OTM in the presence of transactions with each of dependency types.

The performance results obtained with commit-independent subtransactions can be summarized as follows. Enabling global transactions to commit early by issuing retriable subtransactions results in some, although not substantial, improvement in the global throughput of the system. The early committed global transactions do not have a considerable impact on the performance of local transactions because, although some global transactions can commit earlier, their retriable subtransactions continue to contend with local transactions for system resources.

Compensatable subtransactions, on the other hand, do not provide any improvement in the performance of global transactions because commitment of subtransactions earlier than their global parent transactions does not have an effect on the response time of parents. However, early committed compensatable subtransactions were observed to improve the performance of local transactions, as the locks of such subtransactions are released earlier.

REFERENCES

1. R. Agrawal, M. J. Carey, and M. Livny, Concurrency control modeling: Alternatives and implications, *ACM Trans. Database Syst.* 12(4) (1987).
2. P. A. Bernstein, V. Hadzilacos, and N. Goodman, *Concurrency Control and Recovery in Database Systems*, Addison-Wesley, Reading, MA, 1987.
3. Y. Breitbart, H. Garcia-Molina, and A. Silberschatz, Overview of multidatabase transaction management, Tech. Rep. STAN-CS-92-1432, Dept. Computer Science, Stanford University, Stanford, CA, 1992.
4. T. Devirmiş and Ö. Ulusoy, A transaction model for multidatabase systems, in: *2nd Int. Euro-Par Conference*, 1996, pp. 862–865.
5. W. Du, A. Elmagarmid, Y. Leu, and S. Osterman, Effects of local autonomy on heterogeneous distributed database systems, Tech. Rep. TR-ACT-OODS-EI-059-90, Microelectron. and Comput. Corp., Austin, TX, 1990.

6. A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz, A multidatabase transaction model for InterBase, in: *16th Int. Conf. Very Large Databases*, 1990, pp. 507–518.
7. A. Elmagarmid and A. Zhang, A theory of global concurrency control in multidatabase systems, *VLDB J.* 2(3):331–360 (1993).
8. H. Garcia-Molina, Using semantic knowledge for transaction processing in a distributed database, *ACM Trans. Database Syst.* 8(2):186–213 (1983).
9. D. Georgakopoulos, M. Rusinkiewicz, and A. P. Sheth, Using tickets to enforce the serializability of multidatabase transaction, *IEEE Trans. Knowledge and Data Engrg.* 6(1):166–180 (1994).
10. J. Huang, S. Hwang, and J. Srivastava, Concurrency control in federated database systems: A performance study, Tech. Rep. TR93-15, Dept. Computer Science, University of Minnesota, Minneapolis, 1993.
11. W. Litwin, L. Mark, and N. Roussopoulos, Interoperability of multiple autonomous databases, *ACM Computing Surveys* 22(3):267–293 (1990).
12. S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz, A transaction model for multidatabase systems, Tech. Rep. TR-92-14, Dept. Computer Science, University of Texas, Austin, 1992.
13. J. E. Moss, *Nested Transactions: An Approach to Reliable Distributed Computing*, MIT Press, Cambridge, MA, 1985.
14. C. Pu, Superdatabases for composition of heterogeneous databases, in: *4th Int. Conf. Data Engrg.*, 1988.
15. M. Rusinkiewicz, P. Krychniak, and A. Cichocki, Towards a model for multidatabase transactions, Tech. Rep. UH-CS-92-18, Dept. Computer Science, University of Houston, Houston, TX, 1992.
16. H. J. Schek, G. Weikum, and W. Schaad, A multi-level transaction approach to federated dbms transaction management, in: *1st Int. Workshop Interoperability in Multidatabase Syst.*, 1991.
17. H. Schwetman, *CSIM User's Guide*, MCC Tech. Rep. ACT-126-90, 1990.
18. A. Sheth and J. Larson, Federated database systems for managing distributed, heterogeneous, and autonomous databases, *ACM Computing Surveys* 22(3):183–236 (1990).
19. A. Zhang, M. Nodine, B. Bhargava, and O. Bukhres, Ensuring relaxed atomicity for flexible transaction in multidatabase systems, in: *ACM SIGMOD Int. Conf. Management of Data*, 1994, pp. 67–78.

Received 1 April 1996