1993

Decentralized Global Transaction Management in Multidatabase Systems

Aidong Zhang

Jiansan Chen

Ahmed K. Elmagarmid

Purdue University, ake@cs.purdue.edu

Omran Bukhres

Report Number:
93-016
DECENTRALIZED GLOBAL TRANSACTION MANAGEMENT IN MULTIDATABASE SYSTEMS

Aidong Zhang
Jiansan Chen
Ahmed K. Elmagarmid
Omran Bukhres

CSD-TR-93-016
February 1993
Decentralized Global Transaction Management in Multidatabase Systems *

Aidong Zhang, Jiansan Chen, Ahmed K. Elmagarmid, and Omran Bukhres
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907 USA
{azhang, jchen, ake, bukhres}@cs.purdue.edu

Abstract

This paper discusses the implementation of decentralized global transaction management in the multidatabase system environment without violation of local autonomy. The principal concern of this investigation has been to develop a method of global transaction management that is particularly suited to decentralized multidatabase systems. Global concurrency control and atomic commitment have been approached by allowing the serialization and commitment orders of global subtransactions to be determined at each local site, eliminating the need to transfer local information to the global level. Building upon these concepts, we propose a mechanism which supports a decentralized global transaction management approach, completely circumventing the difficulties posed by central coordination.

As multidatabase systems integrate a large number of participating local database systems, a decentralized design is essential to the achievement of a high degree of fault-tolerance.

1 Introduction

A multidatabase system (MDBS) serves to integrate a set of local database systems at various locations (sites). The central concern of such an integration is the preservation of the local autonomy of the component database systems. Aspects of autonomy such as design, execution, and control

*This work was supported by a David Ross Fellowship from the Purdue Research Foundation, a PYI Award from NSF under grant IRI-8857952, and a grant from the Software Engineering Research Center at Purdue University (a National Science Foundation Industry/University Cooperative Research Center — NSF Grant No. ECD-8913133).
have been studied in [GMK88, BS88, DE89, Ve90], and their impact on multidatabase transaction management is discussed in [DEK90, MRKS91, SKS91, MRB+92].

MDBSs process two varieties of transactions. Local transactions access a local database only and are submitted directly to a local database system. Global transactions, on the other hand, may simultaneously access several local databases and are submitted to the global transaction manager (GTM), an important component of an MDBS, superimposed on local database systems, where they are parsed into a set of global subtransactions to be submitted to local database systems. Each local transaction management system preserves the atomicity, isolation, and durability properties [OV91, AA92] of both local and global subtransactions at its site. It is left to the GTM to preserve the atomicity and isolation of global transactions.

The goal of global concurrency control and atomic commitment is to preserve the atomicity and isolation of global transactions. The obstacles to such preservation arise primarily from the constraints posed by the autonomy of local database systems. By definition, a multidatabase system may not have full control over its component database systems, and it must be structured to accommodate the heterogeneity of local database systems. Various potential solutions, both centralized and decentralized in nature, have been proposed in the literature. A centralized approach places the GTM in the hands of a single global coordinator, resulting in a low degree of fault-tolerance and system extensibility. The crash of a single site may block the execution of global transactions at all sites. Centralized management may also complicate the addition to or removal of local systems from the multidatabase system. In contrast, the decentralized approach does not mandate a centralized GTM. A set of GTM servers is distributed to and superimposed upon all local sites, each server having independent control over the execution of the global subtransactions at its site. Each global transaction should be executed independently without a central coordinator.

As a result, the decentralized approach provides a high degree of fault-tolerance, and the system can be easily extended to accommodate new local sites. We therefore anticipate that the decentralized design of global transaction management will become an important feature of multidatabase systems, particularly of those systems integrating a large number of participating local database systems.

Several schemes have been proposed to achieve decentralized transaction management in the MDBS environment. In [BRG92], a decentralized deadlock-free concurrency control method is given for global concurrency control of global transactions. A visible prepare-to-commit state is assumed in [KMS88, BS88, DE89, Vel90, DE90, MRKS91, SKS91, MRB+92].
to be available at each local site. In [VW92], algorithms are presented for the scheduling of global transactions in an MDBS environment in such a manner as to achieve global concurrency control and atomic commitment. These algorithms are based on the assumption that all local database systems maintain rigorous schedules [BGRS91]. A scheme for the implementation of decentralized global transaction management which fully preserves local autonomy has yet to be developed.

In this paper, we investigate the implementation of decentralized global transaction management in the multidatabase system environment without placing any restrictions on local database systems other than local serializability and recoverability [BH87]. A high priority has been placed on the preservation of the heterogeneity of local database systems. In the following sections, the characteristics of a fully decentralized multidatabase architecture are first set forth. We shall then examine the theories advanced regarding global concurrency control and atomic commitment in [ZE93a, ZE93b, EZ93], which indicate that the serialization and commitment orders of global subtransactions can be determined at a purely global level, with no local information being necessary. This approach simplifies the design of the GTM by synchronizing the serialization and commitment orders of global subtransactions at all local sites. As a result, decentralized concurrency control and atomic commitment can be implemented by the GTM servers superimposed on LDBSs, enforcing a globally unique total order on all global transactions.

The remainder of this paper is organized as follows. Section 2 introduces the decentralized system model and the terminology to be employed, while section 3 presents the theoretical bases of decentralized global concurrency control and atomic commitment. In Section 4, a decentralized mechanism is proposed for reliable global transaction management. Concluding remarks are offered in Section 5.

2 Decentralized System Model and Notation

In this section, we will provide a precise definition of the decentralized system under consideration and introduce basic notation and terminology.

2.1 Decentralized System Model

An MDBS consists of a GTM and a set of \( \{ \text{LDBS}_i, \text{for } 1 \leq i \leq m \} \), where each LDBS\(_ i \) is made up of an autonomous database management system on a set \( D_i \) of data items at local site \( \text{LS}_i \). The GTM
is distributed among all machines participating in the MDBS. Each LDBS is associated with a GTM server (GS), and all machines participating in the MDBS can access the GTM interpreter (GI). A global transaction is submitted by invoking a process of the GTM interpreter at the appropriate machine, while a local transaction is submitted directly to a LDBS. All machines are connected by a computer network. Figure 1 illustrates this architecture.

Figure 1: Decentralized multidatabase architecture

The GTM interpreter manages the decomposition and execution of global transactions. In particular, the execution of a global transaction $G_i$ is controlled by the GTM interpreter process $GI_i$, which submits the subtransactions of $G_i$ to the relevant GTM servers for execution. A GTM interpreter process can independently manage the execution of a global transaction without requiring any knowledge of the others’ existence.

A GTM server is responsible for the execution of global subtransactions received from the GTM interpreter processes. It then submits for execution the operations of each global subtransaction to
the LDBS at its associated site. The completion of each operation is acknowledged by the LDBS to the GTM server, which, if necessary, returns these results to the GTM interpreter processes. Each GTM server runs independently from other GTM servers and coordinates only with the GTM interpreter processes from which it receives global subtransactions.

This architecture therefore divides the GTM into two levels, one concerned with the submission of global transactions and the other with the execution of global subtransactions. A GTM interpreter process unilaterally controls the submission of a single global transaction through coordination with the relevant GTM servers, while a GTM server unilaterally controls the executions of global subtransactions at its associated site through coordination with the relevant GTM interpreter processes. Thus, all GTM servers and GTM interpreter processes are independent components capable of making autonomous decisions and a fully decentralized MDBS system is therefore established. The flexibility thus obtained allows users to access the MDBS environment from any machine in the network.

As a necessary assumption of this paper, we presume that the concurrency control and failure recovery mechanisms of LDBSs ensure local serializability\(^1\) and recoverability [BHG87, Had88]. However, no restriction is imposed on these mechanisms.

2.2 Notation

For the elements of a transaction, we assume the availability of four basic operations: \(r(x), w(x), c,\) and \(a,\) where \(c\) and \(a\) are commit and abort termination operations and \(r(x)\) and \(w(x)\) are read and write operations in a local database. Two operations conflict with each other if they access the same data item and at least one of them is a write operation.

A transaction is a partial order of read, write, commit, and abort operations which must specify the order of conflicting operations and which contains exactly one termination operation as the maximum (last) element in the partial order. A more formal definition of a transaction can be found in [BHG87, Had88]. A local transaction is a transaction that accesses the data items at a single local site. A global transaction is a set of global subtransactions where each global subtransaction is a transaction accessing the data items at a single local site. A global transaction \(G_i^{(k)}\) denotes a global subtransaction of \(G_i\) accessing LDBS\(_j\). A global transaction may have more than one global subtransaction at a single local site. A set \(\mathcal{G} = \{G_1, \ldots, G_n\}\) contains those global transactions

\(^1\) In this paper, serializability refers to conflict serializability.
that are submitted to the GTM, and $G_k$ denotes the set of global subtransactions of $G$ at local site $LS_k$. A transaction $T$ refers to either a local or global transaction, while $OP_T$ denotes the set of operations contained in $T$. Two local transactions $T_i$ and $T_j$ conflict, denoted $T_i \prec T_j$, if there exist conflicting operations $o_i$ and $o_j$ such that $o_i \in OP_{T_i}$ and $o_j \in OP_{T_j}$.

Without loss of generality, let global transaction $G_i = \{G_{i1}, G_{i2}, \cdots, G_{im}\}$, where $G_{ij}$ is the global subtransaction at local site $LS_j$. We say that $G_{ij}$ is value-dependent on $G_{ij_1}, \ldots, G_{ij_{i-1}}$ ($1 \leq j_1, \ldots, j_i \leq m$) if the execution of one or more operations in $G_{ij_i}$ is semantically determined by the values read by $G_{ij_1}, \ldots, G_{ij_{i-1}}$.

A schedule over a set of transactions is a partial order of all and only the operations of those transactions which orders all conflicting operations and which respects the order of operations specified by the transactions. A more formal definition of a schedule can be found in [BH87, Had88]. A local schedule $S_k$ is a schedule over both local transactions and global subtransactions which are executed at the local site $LS_k$. A global schedule $S$ is the combination of all local schedules, while a global subschedule $S_G$ is $S$ restricted to the set $G$ of global transactions in $S$.

Beyond the conventional criterion of serializability, we must also define the notion of consistency as it is applied in this paper. Following the traditional approach, a database state is defined as a mapping of every data item to a value of its domain, and the integrity constraints on these data items are used to define database consistency. A database state is considered to be consistent and a schedule to be correct if it preserves these database integrity constraints. In a multidatabase system, there are two types of integrity constraints; local integrity constraints are defined on data items in a single local site, while global integrity constraints are defined on data items in different local sites. Local schedules must preserve local integrity constraints, while global schedules must preserve both local and global integrity constraints.

### 3 Decentralized Theory of Global Transaction Management

In this section, we will present a decentralized theory of concurrency control and atomic commitment for global transaction management in the MDBS environment.
3.1 Theoretical Overview of Global Transaction Management

As an initial condition, we shall permit each global transaction to have more than one global subtransaction at each local site. Such a stipulation results in a greater concurrency of execution of both global subtransactions and local transactions at each local site than would the restriction of each global transaction to have a single subtransaction at each local site. Each global subtransaction must be *locally consistent*; that is, its execution transforms a local database from one local consistent state to another, since the LDBSs treat each global subtransaction as an independent local transaction. In addition, in order to prevent global inconsistency when global subtransactions are interleaved with local transactions, each global subtransaction must be *locally independent*; that is, any two global subtransactions of a single global transaction at a local site must not exchange their data at global level and write simultaneously to the same data items over which a global integrity constraint is defined, since a local transaction which is interleaved between these two global subtransactions may update the data or read inconsistent data from them.

The greatest challenge to the achievement of global concurrency control lies in the determination of the serialization orders of global subtransactions at local sites. The GTM can only control the execution order of global subtransactions by controlling their submissions, while the serialization orders of global subtransactions are controlled by the LDBSs. In [ZE93a, ZE93b], a sufficient condition is proposed for the GTM to determine the serialization order of global subtransactions at local sites. If a set of global subtransactions at a local site is *chain-conflicting*, then the execution order of conflicting operations determines the serialization order of the global subtransactions. A set $G_k = \{G_{1k}, \ldots, G_{mk}\}$ of global subtransactions at local site $LS_k$ is chain-conflicting if there is a total order $G_{i_1 k}, G_{i_2 k}, \ldots, G_{i_m k}$ on $G_k$ such that $G_{i_1 k} \preceq G_{i_2 k} \preceq \ldots \preceq G_{i_m k}$. The conflicting operations of $G_k$ refer to those operations that determine the chain-conflicting relationships of global subtransactions in $G_k$. A global subschedule $S_G$ is *chain-conflicting serializable* if $G$ is chain-conflicting in an order $O$ and $S_G$ is serializable in $O$.

Global serializability cannot be preserved when a global transaction has more than one global subtransaction at a local site. A local transaction may be serialized between two global subtransactions of a single global transaction at a single local site, a situation which is not controllable at the global level.MDBS-serializability, a correctness criterion which is less restrictive than global serializability, is proposed in [EZ93] for global concurrency control. A global schedule $S$ is *MDBS-serializable* if $S$ is serializable without considering the local transactions which are serialized be-
between two global subtransactions that belong to a single global transaction at each local site. If all global subtransactions are locally independent and consistent and the serialization orders of global subtransactions at all local sites are relatively synchronized, then the global schedule is a correct MDBS-serializable schedule. Chain-conflicting serializability can be employed to synchronize the relative serialization orders of global subtransactions at all local sites. Consequently, MDBS-serializability is maintained for global concurrency control of both global and local transactions.

In order to enforce a chain-conflicting relationship on global transactions, an extra operation method is suggested to create chain-conflicting relationships among global subtransactions. Let \(G_{ik}\) and \(G_{jk}\) be nonconflicting global subtransactions at local site \(LS_k\). Conflicts among global transactions can then be simulated. Suppose \(G_{ik}\) is executed before \(G_{jk}\). If \(G_{ik}\) and \(G_{jk}\) do not conflict and an operation of \(G_{ik}\) is on data item \(x\), we then insert operations \(r(x)\) and \(w(x)\) directly before the commit operation of \(G_{jk}\). Let \(G'_{jk}\) denote \(G_{jk}\) after inserting these extra operations. \(G_{ik}\) and \(G'_{jk}\) now conflict with each other, and the effect on \(D_k\) made by \(G'_{jk}\) remains the same as that made by \(G_{jk}\). One advantage of the extra operation method is that it requires nothing from local sites. In addition, the conflict relationships generated by the extra operation method are weaker than those generated by the ticket method [GRS91]; \(G_1 \prec G_2 \prec G_3\) may not imply \(G_1 \prec G_3\).

The goal of atomic commitment is to ensure that either all the global subtransactions of a global transaction commit or none of the effects of each global subtransaction are made permanent. The resolution of local unilateral aborts is the primary consideration in achieving atomic commitment. A local database system that participates in a MDBS environment may unilaterally abort a global subtransaction without agreement from the global level. The sequential commit-retry approach, proposed in [EZ93] to resolve the problem, requires the retrial of any aborted global subtransactions. An aborted global subtransaction is retrievable if it can be resubmitted for execution without creating any inconsistencies in the involved local databases, regardless of whatever operations may have been executed at local sites. To effect a compromise between atomic commitment and concurrency control, the commitment order of global subtransactions must be consistent with their serialization order. If a global subtransaction is aborted and then resubmitted when a global subtransaction is committed or none of the effects of each global subtransaction are made permanent.
initially serialized after it has already committed, the serialization order of global subtransactions may as a result be different from their original serialization order at a given local site. This order, in turn, may be inconsistent with the serialization order of global transactions that all local sites have agreed to enforce.

Another difficulty is encountered when value-dependency relationships are associated with global subtransactions. For instance, let us assume that a value written by $G_{ij}$ at local site $LS_i$ is dependent on a value read by $G_{ij}$ at local site $LS_j$. If $G_{ij}$ commits and $G_{ij}$ aborts, then the retrial of $G_{ij}$ may result in inconsistencies between the data read from the original execution of $G_{ij}$ and from its retrial, since local transactions may be executed after $G_{ij}$ is aborted but before it is retried at $LS_j$. Thus, $G_{ij}$ may not be retriable without violating the value-dependency relationship between $G_{ij}$ and $G_{ij}$. To ensure that each global subtransaction will be retriable, each global subtransaction must commit after all global subtransactions upon which it is value-dependent have committed.

Thus, each global subtransaction remains retriable relative to other global subtransactions belonging to the same global transaction. If the execution of a retried global subtransaction leads to a result which is different from that of its original execution, then those global subtransactions which are value-dependent upon it may be aborted and re-executed. Similarly, each global subtransaction also remains retriable relative to global subtransactions belonging to different global transactions. Those global subtransactions which are serialized after the aborted global subtransaction can be aborted and re-executed in an order which preserves the synchronized relative serialization order of global subtransactions at a local site.

The sequential commit-retry approach achieves the atomic commitment of global transactions provided that no cyclic value-dependency relationships are defined on global subtransactions and each global subtransaction commits after it is retried a sufficient number of times. This is an extension of the retry approach [MRKS91], which, as originally formulated, requires no value-dependencies to be defined on global subtransactions. The sequential commit-retry approach can be combined with the redo and compensation approaches to further extend the applicable global transactions [MRKS91, BGMS92]. Such possibilities will not be addressed here. In general, it has been pointed out in [MEKSA92] that atomic commitment may not be achievable without imposing restrictions on either global transactions or local sites.

As a result, the problem of global concurrency control is reduced to the establishment of syn-

---

4That is, two component global subtransactions of a global transaction may not be mutually value-dependent, either directly or indirectly (through other global subtransactions).
chronization among the relative serialization orders of global subtransactions at all local sites. The atomic commitment of global transactions can be retained by forcing the commitment orders of global subtransactions to follow their value-dependency orders and to be consistent with their serialization orders at local sites. No local information needs to be transferred to the global level, and no cooperation is required between two local sites other than the establishment of a synchronized order among the global subtransactions at all local sites. In the next subsection, we will see that a total order of global transactions can be distributively synchronized at all local sites. In this manner, fully decentralized global concurrency control and atomic commitment can be achieved.

3.2 A Decentralized Approach to Global Transaction Management

The principal issue in the implementation of a decentralized global transaction management scheme based upon the above theory is the synchronization of the relative serialization orders (RSOs) of global subtransactions at all local sites. Our method begins by numbering all GTM servers in an order \( O \) with each GTM server maintaining a site-lock. Prior to executing global transaction \( G_i \), GTM interpreter \( GI_i \) must first request all necessary site-locks from the relevant GTM servers in an order consistent with \( O \). The RSO of \( G_i \) is determined at all relevant sites only when \( GI_i \) has acquired the necessary site-locks. After the RSO of \( G_i \) is determined, \( GI_i \) releases all held site-locks. During this process, if failures occur, \( GI_i \) will request all relevant GTM servers to remove \( G_i \) from the pre-determined RSOs and release all held site-locks. Because the site-locks are requested in an order consistent with \( O \) and the RSO of \( G_i \) is determined only after \( GI_i \) holds all necessary site-locks, the correct synchronization of concurrent site-locks request is ensured and correct RSOs of global transactions at all sites are thus guaranteed. After the RSO of \( G_i \) is determined, \( GI_i \) sends subtransactions of \( G_i \) to the relevant GTM servers. Using the extra operation method, these GTM servers enforce the chain-conflicting relationships and submit the subtransactions for execution according to the pre-determined RSO. A more detailed description of this method is set forth in Section 4.2.

Another crucial issue is the avoidance of cascading aborts. The maintenance of synchronized commitment orders for global subtransactions at all local sites renders them vulnerable to such aborts. Unless global subtransactions are executed serially at each site, the aborting of one subtransaction may cause the aborting of further global subtransactions in an attempt to guarantee a synchronized commitment order of global transactions at all local sites. We here propose a greedy
locking method to prevent cascading aborts which may arise from the concurrent execution of global transactions. This method requires that each GTM server $GS_j$ maintain a dynamic data-lock table which is initially empty. Each entry in the table represents a data-lock for a data item that is currently accessed by global subtransactions. This table is maintained according to the following rules:

- The data-lock requests for each data item are queued and granted in a first-in-first-out manner consistent with the RSO of global subtransactions.

- A global subtransaction can request a sharing data-lock for a data item which it only reads, otherwise, an exclusive data-lock for that data item must be requested.

- A sharing data-lock request for a data item is granted only if it has no data-lock established or if all its existing data-locks are sharing.

- An exclusive data-lock request for a data item is granted only if none of its data-locks exists. This request may be satisfied with a semi-exclusive data-lock if the data item has only sharing data-locks; after all existing sharing data-locks for the data item are released, an exclusive data-lock is granted.

- All the data-locks needed by a global subtransaction must be requested before the execution of the subtransaction. The data-locks held by a global subtransaction are released only when the subtransaction is committed. The released data-locks will be granted accordingly to the relevant subtransactions.

A semi-exclusive lock has the effect of a sharing lock with regard to the lock holder and the effect of exclusive lock with regard to other global subtransactions. That is, read operations on a data item of the subtransaction holding a semi-exclusive lock may be executed, while any upcoming sharing requests for the data are blocked as if the subtransaction held an exclusive lock on the data. Semi-exclusive locks are designed particularly to allow a high degree of concurrency in the preservation of the pre-determined RSO.

A global subtransaction can be executed with currently available data-locks, while an operation of the global subtransaction can be submitted for execution only when the corresponding data-lock has been granted. In this way, the read operations of global subtransactions at each local site are executed concurrently to the greatest possible extent. In addition, since each global subtransaction
releases its data-locks after it has committed, the resubmitted aborted subtransaction can re-use its held data-locks. Moreover, because the global subtransactions that follow an aborted subtransaction $G_{ij}$ in the pre-determined RSO are blocked by $G_{ij}$ through their conflicting operations, the corresponding LDBS cannot decide the serializability orders of these global subtransactions relative to $G_{ij}$ prior to the commitment of $G_{ij}$. The aborting of $G_{ij}$ does not therefore trigger the aborting of any additional global subtransaction and the pre-determined RSO is still preserved. Thus, the greedy locking method prevents cascading aborts while permitting a high degree of concurrency. The following example illustrates the implementation of this method.

**Example 1** Consider three global subtransactions submitted to GTM server $GS_j$ for execution on the local database system $LDBS_j$. $G_{1j}$, $G_{2j}$, and $G_{3j}$ access data defined as \{(R(a, x, w), W(z))\}, \{(R(x, w), W(a, b, y))\}, and \{(R(a, x), W(c, d, y, z))\}, respectively. Assume that the pre-determined RSO is $G_{1j} \rightarrow G_{2j} \rightarrow G_{3j}$. When $G_{1j}$ is executed, the data-lock table is empty, it therefore holds all the necessary data-locks; in this case, sharing data-locks for $a$, $x$, and $w$, and an exclusive data-lock for $z$. $G_{2j}$ is then submitted and is granted exclusive data-locks for $b$ and $y$, sharing data-locks for $x$ and $w$, and a semi-exclusive data-lock for $a$. If the execution involves a $w(a)$ operation, $G_{2j}$ will be blocked until the commitment of $G_{1j}$; $G_{2j}$, however, can be proceeded to execute operations $r(a)$, $r(x)$, and $r(w)$ simultaneously with $G_{1j}$. By the same token, while $G_{3j}$ can be submitted for execution with a sharing data-lock for $x$ and exclusive data-locks for $c$ and $d$, its execution will be blocked either by $z$ until $G_{1j}$ is committed or by $a$ or $y$ until $G_{2j}$ is committed.

In this example, we see that both $G_{1j}$ and $G_{3j}$ read $a$, and $G_{2j}$ may read/write $a$. Under the terms of a semi-exclusive lock, the reading of $a$ is shared by $G_{1j}$ and $G_{2j}$, while $G_{2j}$ is blocked by its $w(a)$ operation and $G_{3j}$ is blocked by its $r(a)$ and $w(a)$ operations. As all three subtransactions hold a sharing data-lock for $x$, the $r(x)$ operations in the three subtransactions can all be performed simultaneously. A high degree of concurrency is thus achieved.

Assume that $G_{1j}$ is aborted by $LDBS_j$. Because $G_{2j}$ is blocked on the first operation that may conflict with an operation of $G_{1j}$ (e.g., $w(a)$), $LDBS_j$ cannot yet arrange the RSO of $G_{1j}$ and $G_{2j}$. A similar situation may also arise regarding the RSO of $G_{1j}$ and $G_{3j}$. Cascading aborts that might be caused by the aborting of $G_{1j}$ are thus avoided. $G_{1j}$ can therefore be resubmitted for execution with its data-locks and the pre-determined RSO can still be enforced. After $G_{1j}$ is committed, its data-locks are released, allowing $G_{2j}$ to proceed, its serializability order relative to

---

$^5$R(...) consists of read-only data, while W(...) consists of other varieties of data.
Glj is then determined by LDBSj.

4 A Decentralized Global Transaction Management Mechanism

In this section, we illustrate the incorporation of the developed approaches into a decentralized global transaction mechanism. A global transaction model for the specification of global transactions is first presented. The concurrency control and atomic commitment of global transactions can then be easily enforced through the synchronized execution of the conflict and commit operations of global transactions at all local sites.

4.1 Global Transaction Model

The proposed theory requires that no cyclic value-dependency relationships be defined on global transactions submitted to the GTM. Such global transactions are termed acyclic global transactions. As the proposed theory does permit multiple global subtransactions of a global transaction to be executed at a local site, users may conveniently specify acyclic global transactions. If two global subtransactions are cyclically value-dependent upon each other, then their decomposition into additional global subtransactions may break the cycle of value-dependency. For this purpose, the global transaction model must provide users fine control over the specification of the decomposition of global transactions. This is accomplished by extending the conventional transaction model to allow users to insert breakpoints [GM83, FO89] in global transactions, directing the GTM in the decomposition of global transactions before they are submitted to local sites.

The feasibility of such decomposition depends upon the semantics of the global transactions. Each decomposed global subtransaction must be locally independent and consistent. As breakpoints can be removed from global transactions after global-level decomposition is complete, decomposed global subtransactions contain no breakpoints. There are therefore no special requirements made of the transaction models used at local sites, and the heterogeneity of local database systems can still be assumed.

4.2 A Decentralized Global Transaction Management Algorithm

Our decentralized global transaction management algorithm incorporates the approaches of the decentralized concurrency control and atomic commitment proposed in Section 3. The execution
of a global transaction $G_i$ consists of three phases:

**Phase 1: The determination of relative serializability orders**

This phase determines the relative serializability orders (RSOs) of global transactions at local sites, which are activated when the GTM interpreter processes responsible for executing global transactions submit global subtransactions to the GTM servers for execution. Each GTM server maintains a site-lock. Let $O$ be an order on all GTM servers. GTM interpreter $GI_i$, which executes $G_i$, must request the necessary site-locks from the relevant GTM servers in an order consistent with $O$ before submitting the subtransactions of $G_i$. Site-locks are allocated according to the following rules:

- All site-lock requests received by a GTM server $GS_j$ which is associated with site $LS_j$ are handled in a first-in-first-out fashion; $GS_j$ can process and grant a site-lock request only when its site-lock is available.

- $GI_i$ must be blocked when its current requested site-lock is not available. $GI_i$ submits all its global subtransactions accessing $LS_j$ to $GS_j$ when the site-lock is granted from $GS_j$. $GI_i$ can then send the next site-lock request to the relevant GTM server.

- $GI_i$ releases all held site-locks after all its global subtransactions are submitted and cannot request any further site-locks.

At each site, the RSOs of the global subtransactions of different global transactions are determined by their site-lock granting orders, while the RSOs of the global subtransactions of a global transaction at a local site are determined by the semantics of the global subtransactions. This method of ordering is deadlock-free and totally distributed.

**Phase 2: The execution of global subtransactions**

The execution of global subtransactions at each GTM server is invoked by the pre-determined RSO. To implement the extra operation method described in Section 3.1, before it is invoked, each global subtransaction $G_{ij}$ has operations $r(x)w(x)$ been inserted directly before its commit operation. Here $z$ is a data item accessed by the global subtransaction immediately preceding $G_{ij}$ in the pre-determined RSO, if such a global subtransaction exists. The execution of $G_{ij}$ is carried out by GTM server process $SP_{ij}$ created by GTM server $GS_j$ and must obey both the rules of the greedy
locking method described in Section 3.2 for the request of data-locks and the following additional stipulations:

- An operation of $G_{ij}$ is submitted for execution when the corresponding data-lock is granted.
- When an operation is completed, $SP_{ij}$ sends the result to GTM interpreter $GI_i$, if necessary; $GI_i$, in turn, sends the result to the GTM servers associated with value-dependency-related global subtransactions. If the data for an operation is unavailable, the execution is blocked by the data.
- When $SP_{ij}$ reaches a commit operation, it sends a commit request to both $GS_j$ and $GI_i$. $SP_{ij}$ can commit $G_{ij}$ only upon receiving approval from both $GS_j$ and $GI_i$.
- When $SP_{ij}$ executes an abort operation, it reports the abort to both $GS_j$ and $GI_i$.

Phase 3: Commitment control of global subtransactions

The control of commitment of global subtransactions is governed by the following rules:

- When receiving a commit request from $SP_{ij}$ for $G_{ij}$, GTM interpreter $GI_i$, which executes $G_i$, verifies whether all global subtransactions upon which $G_{ij}$ is value-dependent have committed. If this is the case, $GI_i$ sends its approval to $GS_j$. GTM server $GS_j$ also approves the commitment of $G_{ij}$ when all global subtransactions that precede $G_{ij}$ in the pre-determined RSO have committed.
- When a subtransaction $G_{ij}$ is aborted, $GI_i$ then sends abort commands to all subtransactions of $G_i$ that are directly or indirectly value-dependent upon $G_{ij}$; these subtransactions then abort themselves accordingly. $GI_i$ then re-executes the aborted subtransactions, beginning from phase 2.

This algorithm allows transaction management decisions concerning a global transaction $G_i$ to be made independently by the individual GTM servers that execute the subtransactions of $G_i$ and by the GTM interpreter that executes $G_i$ based on locally available or coordinating information. This algorithm is therefore fully decentralized, in that each global transaction can run independently, requiring no knowledge of other global transactions. The GTM can then be distributed among the machines from which global transactions are issued, resulting in an approach which is both flexible and reliable.
5 Conclusions

In this paper, we have presented a fully decentralized multidatabase system architecture, a decentralized theory of global concurrency control and atomic commitment, and a mechanism for supporting decentralized global transaction management. This architecture supports remote access to the entire MDBS environment from all machines in the computer network. The decentralized theory of global transaction management provides the theoretical basis for the proposed architecture, while the mechanism provided integrates the architecture with its supporting theory.

We have thus demonstrated the potential advantages realizable with the implementation of decentralized multidatabase systems. The pitfalls of centralized coordination can be completely avoided, making possible the development of multidatabase systems with improved fault-tolerance and system extensibility. The advantages of this model will become more evident in an environment with large number of participating sites.

The design and implementation of the proposed mechanism is currently being investigated as part of the InterBase project at Purdue University.

References


