7

Optimistic Concurrency Control for Maintaining the Global Integrity Constraint in MDBSs

Kyuwoong Lee and Seog Park
Database Research Lab., Dept. of CS., Sogang Univ.
Seoul, C.P.O Box 1142, KOREA
{qlee, spark}@dblab.sogang.ac.kr

Abstract

The multidatabase system(MDBS) has a global database, a set of global and local transactions, and a global transaction manager(GTM) which is built on top of a number of pre-existing database management systems(DBMS) that are being integrated into a single MDBS. The global transaction manager has the responsibility for maintaining the global consistency of MDBS. It is impossible for LDBSs to preserve these global integrity constraints because neither the local user nor the transaction manager of each LDBS is aware of the integration process and these integrity constraints. Locally consistent transactions may generate global inconsistencies with the existence of global integrity constraints. Furthermore, the global serializability may be violated, even though each the local schedule is serializable. Hence, we need the global transaction management method that ensure the global serializability and logical consistency, together. In this paper, we investigate the transaction model for maintaining the global integrity constraints and propose the optimistic concurrency control method to guarantee the global serializability and logical consistency based on our transaction model.

Keywords

Global and Local Transaction, Concurrency Control, Integrity Constraint, Multidatabase System, Consistency

1 INTRODUCTION

The multidatabase system has a global database, a set of global and local transactions, and a global transaction manager(GTM) which is built on top of a number of pre-existing database management systems(DBMS) that are being integrated into a single MDBS. Since the integration of the various DBMSs into a MDBS results in the introduction of inter-site constraints, we have to determine where and how the global integrity constraint is maintained.

Integrity and Internal Control in Information Systems Volume 1
S. Jajodia, W. List, G. McGregor & L. Strous (Eds) © 1997 IFIP. Published by Chapman & Hall

The LDBS is not proper to keep up the global integrity constraint because it is unaware of global constraints as well as the information of data at different site. The global transaction manager has the responsibility for maintaining the global consistency of MDBS. In order to keep the global consistency, the global schedule that is combined in each local schedule must be globally serializable, and its database state must be satisfied with given global integrity constraints. Each pre-existing local DBMS defines certain integrity constraints among the data items at located the single site. However, as a number of DBMSs are integrated into an MDBS, certain global consistency requirements or integrity constraints are required on the distributed data. These distributed integrity constraints arise naturally whenever the data that is semantically related is stored in different local database systems. The global integrity constraints is to describe the integrity constraints associated with data items in different local databases, which specify the global configuration of the data that are considered semantically correct.

Locally consistent transactions may generate global inconsistencies with the existence of global integrity constraints. When an update on data that is semantically related with the data at different site is executed under the control of LDBS, the global inconsistent state may be generated. We show an example that locally consistent transaction produces the global inconsistent state.

Example 1 *Consider an Employee-Department database with three relations:*
EMP(EMP♯, ENAME, DEPT♯, SAL)
DEPT(DEPT♯, DNAME)
PROJ(PROJ♯, PNAME, EMP♯)
We assume that the global database consists of two sites S_x with a relation EMP and S_y with two relations DEPT and PROJ, as shown in Figure 1. Each LDBS has pre-existing integrity constraints as followings.
Local Integrity Constraints LIC_1: $(EMP.SAL < 200)$
Local Integrity Constraints LIC_2: $(DEPT.DEPT\sharp < 100)$
In addition, the global integrity constraints is defined over two sites as following.
Global Integrity Constraints GIC_1:
$(EMP.DEPT\sharp \in DEPT.DEPT\sharp)$ *and*
$(PROJ.EMP\sharp \in EMP.EMP\sharp)$
We also assume that there are two global transactions and a local transaction. Global transaction G_1 reads tuple E1 of relation EMP and updates tuple D1 of relation DEPT. Global transaction G_2 updates tuples E1 and P1 of relation EMP and PROJ, respectively. Local transaction L_1 updates tuple P1 of PROJ and reads tuple D1 of relation DEPT at site S_y. At each LDBS, two global transactions and a local transaction are executed as following schedules.
Site S_x : $R_{G1}(E1)W_{G2}(E1)$
Site S_y : $W_{G2}(P1)W_{L1}(P1)R_{L1}(D1)W_{G1}(D1)$

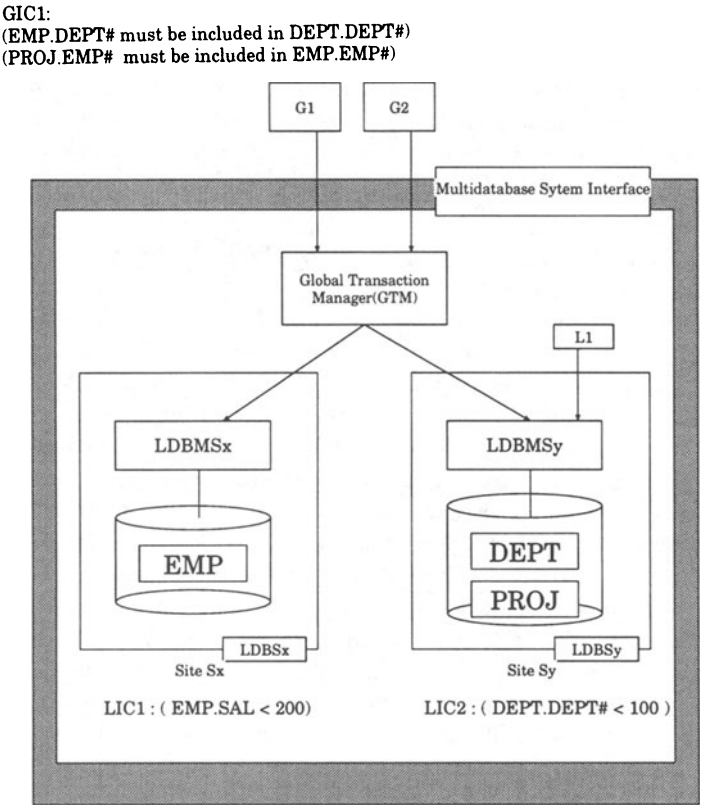

Figure 1 Global Integrity Constraints

Each schedule produces locally consistent state and local integrity constraints LIC_1 and LIC_2 are maintained by each LDBS. Hence, schedules locally have the execution consistency and logical consistency.

However, in this example, if the local transaction updates the tuple P1 with invalid EMP♯ value that is not included relation EMP, locally consistent schedule generates global logical inconsistency *that violates the global integrity constraint GIC_1. The LDBS cannot determine whether the EMP♯ value of tuple P1 in relation PROJ is valid or not, because the LDBS cannot access the relation EMP. Since the verifying the global integrity constraints can be processed only by they global transaction manager, we need the restriction that the local transaction cannot update the data item which is defined in the global integrity constraint. It is nature that the global inconsistency that caused by local transaction is generated because the LDBS controls local transactions without the knowledge of global integrity constraints. Also, these global inconsistencies cannot be easily detected by global transaction manager, because the global transaction manager is unaware of existence of local transaction.*

Moreover, schedules violate the global serializability that is used as correctness criteria for execution consistency *in most database systems. Although each schedule is locally conflict serializable, entire schedule is not globally conflict serializable because there is the cyclic global serialization order* $G_1 \rightarrow L_1 \rightarrow G_2 \rightarrow G_1$. *Therefore, the transaction execution in this example cannot ensure the global serializability, as well as the logical consistency.* □

As illustrated in Example 1, if there exists global integrity constraints between distributed data items, an update that is executed locally may produce the global inconsistency. In order to maintain the global consistent database, the GTM has to always monitor the global database state whether it is satisfied with given global integrity constraints It requires highly expensive cost of MDBS. We may also need to temporarily tolerate inconsistencies among related data while the GTM successfully executes the update propagation procedure according to global integrity constraints. Hence, it is not realistic to maintain the global consistency of MDBS.

Some solutions have been proposed to manage the distributed integrity constraints [11, 5, 17, 4]. These methods are mainly contribute to check distributed constraints after the execution of transactions or describe the formal notation of constraints under the assumption that the GTM that ensures the global serializability. Other approaches also have been proposed to ensure the global serializability for execution consistency of concurrent transaction [13, 16, 18, 14, 9, 10, 3, 6]. However, these methods did not consider the execution and logical consistency together. In this paper, we investigate the transaction model for maintaining the global integrity constraints and propose the concurrency control method to guarantee the global serializability and logical consistency based on our transaction model.

The rest of the paper is as following. In Section 2, we discuss the global integrity constraints and then propose the transaction model that supports the global integrity constraints. In Section 3, we introduce the optimistic concurrency control for multidatabase transaction management to serialize the indirect conflict operation as well as direct conflict operation. In Section 4, we prove the correctness of our proposed concurrency control and also evaluate the performance by simulation experiments in Section 5. We conclude the paper in Section 6.

2 MULTIDATABASE TRANSACTION MODEL

2.1 Integrity Constraints and Transaction Model

The most fundamental issue of global integrity constraints is how and where the global integrity constraints is maintained without the violation of local autonomy. As illustrated in Section 1, the GTM must examine whether the

value of global data item is consistent according to the global integrity constraint. Since the LDBS does not have the capability to maintain the global integrity constraint, the data item which is defined in the global integrity constraint should be managed by the GTM.

The introduction of inter-site constraints enable us to partition the set of data item at a site, D_i, into local data items, LD_i, and global data items, GD_i, such that $LD_i \cap GD_i = \phi$ and $D_i = LD_i \cup GD_i$[14]. Furthermore, if there is an integrity constraints between the data item $d_i \in D_i$ and $d_j \in D_j$, $i \neq j$, then data items d_i and d_j are global data items in GD_i and GD_j, respectively. Therefore, we partition the data items of MDBS in two groups, as following.

global data item :
the set of data item which is defined in the global integrity constraints
local data item :
the set of data item which is defined in the local integrity constraints

In a MDBS, pre-existing local applications can be assumed to be satisfied with the local database integrity constraints. However, they are unaware of the global integrity constraints. It is clear that the global database state may be inconsistent if the local transaction updates the global data without the knowledge of global integrity constraints and control of MDBS.

To avoid such global inconsistency, the GTM must have the responsibility for preserving the global consistency. Hence, restriction on the local transaction is necessitated in MDBS. In our work, we prohibit the local transaction from updating on global data item without the knowledge of global integrity constraints. This restriction enable us to easily maintain the global integrity constraints and reduce the cost of verifying the global consistency. However, the local transaction is not restricted to read the data item. The local transaction can read both local and global data items. Some researches propose the method that the global transaction is restricted to read and update the data item and they assume that there is no inter-site constraints [6, 7, 14, 15]. If the global transaction is restricted to read or update or there is no global integrity constrains, we loss the meaning of MDBS. The major purpose of MDBS is to access the distributed data. If the global transaction is restricted to access some data, it is not realistic. In practical cases, the global transaction cannot be imposed on updating the data item, as well as reading the data item.

In our work, global transaction is free to access data items. Our multidatabase transaction model for maintaining global integrity constraints prohibit only the local transaction from writing the global data item, as illustrated in Table 1. We denote multidatabase transaction model in Table 1 as *the Global-Free transaction model* for MDBSs. In following sections, we investigate the type of conflicts and its possibility based on out *Global-Free* transaction model.

Table 1 Global-Free Transaction Model for MDBS

Transaction	Data	Data Item	
		Local Data	Global Data
Local Transaction	Read Operation	○	○
	Write Operation	○	×
Global Transaction	Read Operation	○	○
	Write Operation	○	○

○ : possible × : impossible

2.2 Direct Conflict Operations between Local and Global Transaction

In a MDBS, the serializability of local schedule is not sufficient to keep the global serializability. The MDBS needs to know the local schedules to assure that the global consistency is not violated. Its basic problem is to resolve the indirect conflict that is caused by unknown local transaction. The MDBS, therefore, must deal with not only direct conflict between global transactions but also the indirect conflict that is caused by the local transaction. The basics of indirect conflict problem and local autonomy is described in many researches[14, 6, 10, 12, 16, 13]. We briefly examine the notion of indirect conflict problem between global transactions.

Definition 1 *The global transaction G_i and G_j are in indirect conflict in schedule S if and only if there is a local transaction sequence $L_1, L_2, \ldots, L_r$, such that G_i is in direct conflict with L_1 , L_1 is in direct conflict with L_2, ..., finally, L_r is in direct conflict with G_j.* □

We denote D(G_i) as the set of data items that is accessed by the transaction G_i. If two transactions G_i and G_j are in indirect conflict, then either $D(G_i) \cap D(G_j) = \phi$ or they have the shared data items that are accessed by only read operations of both G_i and G_j. Otherwise, they have the direct conflict relationship. We describe the procedure to find the type of conflict between given two global transactions at a site. Let the $Read\_Set(G_i)$ be set of data item that is read by global transaction G_i and $Write\_Set(G_i)$ be set of data item that is written by global transaction G_i.

If $(Read\_Set(G_i) \cap Write\_Set(G_j) \neq \phi$) or
$(Write\_Set(G_i) \cap \{Write\_Set(G_j) \cup Read\_Set(G_j)\} \neq \phi$) then
G_i and G_j are in *direct conflict*

else
    if (there exists the set of local transaction that
        makes the global transaction have a serialization order)
            G_i and G_j are in *indirect conflict*
    endif
endif

In this procedure, GTM cannot find the set of local transaction that causes the serialization order between given global transactions because of local autonomy. A indirect conflict consists of at least two direct conflicts between local and global transaction. If we can prevent one of them, indirect conflict cannot occur. For example, We consider the following schedule H_{x1} at site S_x.

$$H_{x1} : R_{Gix}(a)W_{Lx}(a)R_{Lx}(b)W_{Gjx}(b)$$

The operation $R_{Gix}(a)$ of G_{ix} is direct conflict with $W_{Lx}(a)$ of L_x and the operation $W_{Gjx}(b)$ of G_{jx} is also direct conflict with $R_{Lx}(b)$ of L_x. A global subtransaction G_{ix} are indirect conflict with G_{jx}, since the local transaction L_x has direct conflict with both G_{ix} and G_{jx} at a site S_x. In this schedule, if we can prevent one of two direct conflicts, the indirect conflict between G_i and G_j cannot occur.

Hence, we need to investigate the direct conflict between local and global transaction. Table 2 shows the direct conflict between local and global transaction based on our *Global-Free Transaction Model.* In Table 2, "○" means

Table 2 Direct Conflict between the Local and Global Transaction

			Global Transaction			
			Read Operation		Write Operation	
			Global data	Local data	Global data	Local data
Local Transaction	Read Operation	Global data	○	○	×	○
		Local data	○	○	○	×
	Write Operation	Local data	○	×	○	×

○ : no direct conflict × : direct conflict

that there cannot exist direct conflict between the local and global transaction and "×" means that there can be the direct conflict. In above schedule

H_{x1}, we assume that the data item a is a local data and b is a global data. According to the Table 2, two operations $R_{Gix}(a)$ and $W_{Lx}(a)$ are in direct conflict because the local transaction write the local data a and global transaction read the same data. Similarly, there exists the direct conflict on the global data b between $R_{Lx}(b)$ and $W_{Gjx}(b)$. Hence, two operations $R_{Gix}(a)$ and $W_{Gjx}(b)$ are in indirect conflict.

In this schedule H_{x1}, two operations, $R_{Gix}(a)$ and $W_{Gjx}(b)$, are *indirect conflicting operations.*

Definition 2 *The indirect conflicting operation is the operation of global transaction that causes the indirect conflict with the other global transaction. Formally speaking, if the operation $p_i(x)$ of global transaction G_i is indirect conflict with the operation $q_j(y)$ of other global transaction G_j, $i \neq j$, then two operations $p_i(x)$ and $q_j(y)$ are indirect conflicting operations. It is not necessary that two data items, x and y, are distinct. If two operations $p_i(x)$ is indirect conflict with $q_j(y)$, $x = y$, then operations p and q must be both read operations. Otherwise, two operations $p_i(x)$ and $q_j(y)$ are direct conflicting operations.* □

However, we consider following schedule H_{x2}. The schedule H_{x2} is the same as H_{x1} except the operation $R_{Gjx}(b)$.

$$H_{x2} : R_{Gix}(a)W_{Lx}(a)R_{Lx}(b)R_{Gjx}(b)$$

In schedule H_{x2}, since the local transaction reads the global data item b and the global transaction G_{jx} also reads it, there cannot exist the *direct conflict.* The global transaction that reads the global data item cannot has direct conflict with a local transaction. Hence, there cannot exist the *indirect conflict* between G_{ix} and G_{jx}. In the most of proposed researches[8, 9, 10], however, G_{ix} and G_{jx} are forced to have intentional direct conflict between them. Such a forced direct conflict between global transaction causes to reduce the concurrency degree.

But, the Table2 cannot be not directly used to the GTM, because the GTM cannot take any kind of information from LDBS. We abstract the Table2 to deduce the possibility of indirect conflict between global transactions.

3 OPTIMISTIC GLOBAL TRANSACTION MANAGER

3.1 Management of Indirect Conflict

The GTM cannot know which type of local transaction is executed in the site, when the global subtransaction is submitted to the LDBMS in that site. Hence, we need the global transaction manager that resolves the indirect conflict by controlling only global transactions.

Our optimistic global transaction manager validates that a global transac-

tion has the resolvable direct and indirect conflict after its execution at all accessed sites. The indirect conflict has at least two direct conflict between local and global transaction. As illustrated in Table 2, the read operation for global data item in global transaction cannot make the direct conflict with any local transaction. We can inference the possibility of indirect conflict between global transactions based on Table 2. Hence, we know whether the indirect conflict can exist or not without any information of local transaction, as described in Table 3. Since the read operation for global data item of global

Table 3 Indirect Conflicts between Global Transactions

Global Transaction G_i		Read		Write
Global Transaction G_j		Global Data	Local Data	
Read	Global Data	○	○	○
	Local Data	○	×	×
Write		○	×	×

○ : no indirect conflict × : indirect conflict can occur

transaction cannot make any conflict with global and local transaction in Table 3, we can easily ensure the global serializability. In other cases, we cannot ensure the global serializability.

We define the *stable global transaction* according to the Table 3. We use $Site_{Gi}(RL)$ to denote the set of sites in which the global transaction G_i reads the local data item and $Site_{Gi}(W)$ to denote the set of sites in which the G_i writes the data item. But, we do not consider the site that G_i is direct conflict with other global transactions.

Definition 3 *A global transaction G_i is stable global transaction, if and only if, for every other global transaction G_j that accesses the same site as G_i accesses,*

$$Site_{Gi}(RL) \cap \{Site_{Gj}(RL) \cup Site_{Gj}(W) = \phi\} \text{ and}$$
$$Site_{Gi}(W) \cap \{Site_{Gj}(RL) \cup Site_{Gj}(W) = \phi\} \square$$

For example, in the schedule H_{x1} of previous section, the global transaction G_{ix} is *not* a *stable global transaction*, because $Site_{Gix}(RL) \cap Site_{Gjx}(W) \neq \phi$. A global transaction that is not *stable* may have the indirect conflict with other global transactions. In the schedule H_{x2}, however, G_i is *stable global transaction.*

3.2 Management of Direct Conflict

The GTM should resolve the direct conflict between global transactions, in addition to the indirect conflict. To serialize the direct conflicting operation, the GTM maintain the Global Transaction Serialization Graph($GTSG$). The directed GTSG, denoted $GTSG(V, E)$ consists of the set of nodes, V, and set of directed edges, E, such that V represents the global transaction and E represents serialization order of global transaction. The set V contains the recently committed global transaction and all active global transactions which are not yet committed. A directed edge $E(G_i, G_j)$ represents that a operation $p_{ix}(a)$ of global transaction G_i precedes a operation $q_{jx}(a)$ of global transaction G_j at a site x and operation p is direct conflict with operation q.

The procedure for resolving the direct conflict is the almost same as traditional *Serialization Graph Testing* (SGT)[2]. In order to have the same relative serialization order of direct conflicting operations in their corresponding LDBSs, the GTM must delay $p_{ix}(a)$ until the GTM acknowledges all direct conflicting operations. This handshake method can be implemented as in conventional Time-Stamp Ordering Method and Serialization Graph Testing Method[2]. If all subtransactions of global transaction G_i enter their prepared-to-commit state, the GTM checks if the GTSG contains a cycle. If the GTSG has a cycle, the resulting schedule would be non-serializable schedule. Therefore, the global transaction is aborted and then should be restarted. Otherwise, the global transaction is successfully validated for direct conflict.

3.3 Validation Procedure

In previous subsections, we describe the method to validate the indirect conflicting operations as well as direct conflicting operations. We present the overall validation procedure of GTM for multidatabase transaction management. First, our validation procedure verifies that there is no cycle in the GTSG, and then validates there is no cycle in the GTSG.

The overall validation procedure is described as following. When the optimistic scheduler receives an operation $p_{ix}(a)$ of global transaction G_i from the GTM, it creates a node for G_i in GTSG, if one does not exist. Then it adds an edge from G_j to G_i for every previously scheduled operation $q_{jx}(a)$ that direct conflicts with $p_{ix}(a)$. After all the direct conflicting operations $q_{jx}(a)$ have been successfully completed, the operation $p_{ix}(a)$ is sent to the corresponding site x. If all subtransactions of global transaction G_i enter their prepared-to-commit state, the global transaction enters its validation phase. The validation phase is divided into two parts. Fist, the GTM validates G_i can be conflict serializable with all transactions in GTSG. It is performed by testing that the GTSG is cyclic, as described in Section 3.2. And then the GTM checks whether the G_i can have the indirect conflict with respect to all

transactions in GTSG. It is performed by testing the ***stability*** , as defined in Definition 3. If the GTM validates that the G_i is *stable global transaction*, G_i is guaranteed not to be indirect conflict with any other global transactions. If all of validation phase is successfully completed, G_i can be safely committed. However, if any part of validation phase is failed, G_i may cause the global inconsistency. G_i is aborted and restarted.

Overall Validation Procedure

Step 1 : Adding a node and its edge to GTSG
When an operation $p_i(a)$ of global transaction G_i is received from GTM
if a node for G_i does not exist in GTSG
Create a node for G_i in GTSG
for each previously scheduled operation $q_j(a)$ of global transaction G_j, which is direct conflict with $p_i(a)$, $i \neq j$
Add an edge from G_j to G_i

Step 2 : Submitting the operation
Submit the $p_i(a)$ to its corresponding LDBS if all $q_j(a)$ have been successfully completed.

Step 3 : Validating the global transaction for direct and indirect conflicting operation
If all subtransaction of G_i enter their prepared-to-commit state
if current GTSG has no cycle
/* G_i is not direct conflict with other global transactions */
if all nodes G_j in GTSG are not *indirect conflictable* with G_i
Commit G_i
Exit procedure
endif
else
Remove a node for G_i with all incident edges
Restart G_i
endif
endif

Example 2 *We illustrate the example that the global serializability and logical consistency are guaranteed by our proposed optimistic protocol. We assume that there are two sites and each site produces the following local schedules* H_x *and* H_y.

$$H_x : W_{Gix}(a)W_{Gjx}(a)$$
$$H_y : W_{Gjy}(c)R_{Ly}(c)R_{Ly}(b)R_{Giy}(b)$$

We also assume that there is the global integrity constraints $(b > a)$ *between data items that are distributed. Hence,* b *is global data item and* a *and* c *are local data items.*

At site S_x, since there is direct conflict between G_i and G_j, there is the local serialization order $G_i \rightarrow G_j$. At site S_y, however, any direct conflict does not exist. According to the step 1 and 2, *GTSG is constructed and edge from $G_i \rightarrow G_j$ is inserted. All operations are submitted at corresponding site and each LDBS produces the schedule H_x and H_y, respectively. At this time, the G_i is assumed to enter its validation phase in* step 3 *of above procedure.*

Since there is no cycle in GTSG, the G_i can pass the validation test for direct conflict. It means that the G_i is conflict serializable with respect to other global transactions. And then, the validation test for indirect conflict is performed. There is no intersection set which is defined in Definition 3. Hence, G_i is not indirect conflictable *with the G_j. Finally, the G_i can globally commit. At later, similarly, the G_j enters its validation phase and can commit successfully.*

Other methods[8, 9, 10] that force the G_i and G_j to have the intentional conflict between them should abort one of them, in order to ensure the global serializability. Therefore, our proposed optimistic method can get the more concurrency degree than other methods that have the intentional conflict. □

4 CORRECTNESS OF OPTIMISTIC CONCURRENCY CONTROL

In order to ensure the global serializability, we have to serialize the indirect and direct conflicting operations. Hence, for the direct conflicting operation between global transactions, proposed method guarantees the *conflict serializability* by using the acyclic GTSG, and for the indirect conflict operation, our method ensures that *stable transaction* cannot have any indirect conflict. We prove that an acyclic GTSG is conflict serializable and also prove that a *stable* global transaction cannot have any indirect conflict. Finally, we show that a schedule produced by proposed method is globally serializable.

Lemma 1 *If a global transaction G_i is stable global transaction, G_i cannot have any indirect conflict.*

Proof. For all global transactions G_j that share the site with a stable global transaction G_i, only following schedules are possible according to the Definition 3.

CASE 1 : two operations are both read operations
$R_{Gi}(a)$ $R_{Gj}(b)$
/* One of data items a and b must be a global data item and another one can be a global or local data item. The execution order of two operation can be reversed. If both a and b are global data, they can be the same data item. Otherwise, $a \neq b$ because $GlobalData \cap LocalData = \phi$ in our

Global-Free transaction model. */

CASE 2 : One operation is read operation and another one is write operation
$R_{Gi}(a)$ $W_{Gj}(b)$ or $W_{Gi}(b)$ $R_{Gj}(a)$
/* The data item a must be a global data item and b can be the global or local data item. The execution order of two operation can be reversed. In this case, a and b are necessarily distinct. */

To show that there is no indirect conflict with *stable* global transaction, without the loss of generality, we assume that the *stable* global transaction G_i has the indirect conflict with G_j. Since the G_i and G_j are in indirect conflict, each G_i and G_j must have at least one direct conflict with a local transaction.

In CASE 1, G_i and G_j are direct conflict with a local transaction, respectively, only if there are write operations $W_L(a)$ and $W_L(b)$. It means that at least one of write operations writes on the global data item. However, in our *Global-Free* transaction model, the local transaction is not allowed to write on the global data item. Similarly, in CASE 2, each G_i and G_j must have a direct conflict with a local transaction, since we assume that G_i and G_j are in indirect conflict. The local transaction must write on the global data item in order to make the direct conflict with a global transaction in CASE 2.

Therefore, all of CASE 1 and 2, *stable* global transaction have indirect conflict only if the local transaction must write on the global data item. In *Global-Free* transaction model, the local transaction is not allowed to write on the global data item. Hence, It is contradict to our assumption that *stable* global transaction has the indirect conflict. Finally, *stable* global transaction cannot have any indirect conflict. □

Lemma 2 *If there is no cycle in the GTSG for the schedule S, it is conflict serializable.*

Proof. For a given schedule S, if the traditional serialization graph SG has no cycle, it is trivial that the schedule S is conflict serializable. It is proved in many literatures [2]. The GTSG is almost the same as the traditional SG except that GTSG consists of only global transactions. Hence, if there is no cycle in GTST, the schedule S is conflict serializable. □

Theorem 1 *The proposed optimistic concurrency control method produces the globally serializable schedule.*

Proof. According to our optimistic concurrency control, the global transaction can be committed if the GTSG has no cycle and the global transaction is a *stable* global transaction. If the serialization graph has no cycle, then we can guarantee that the schedule is serializable. Since the GTSG consists of only

the global transaction, if it has no cycle, it is serializable with respect to global transactions(Lemma2). A *stable* global transaction is ensured not to have any indirect conflict with other global transactions (Lemma1). Thus, the global transaction that is validated by our optimistic concurrency control can be serializable with respect to other global transactions and it cannot make any additional serialization order with local transactions. Hence, the execution schedule that is produced by our method is globally serializable. □

5 EVALUATION OF OPTIMISTIC CONCURRENCY CONTROL PROTOCOL

In this section, we show the experimental result of performance evaluation. Particularly, we compare our simulation result to *Optimistic Ticket Method*(OTM) [8, 9, 10]. The OTM protocol is one of the most widely used protocol for MDBS transaction management. Since the basic idea of OTM is very simple and accurate, it can be easily adopted to various systems. The overall simulation model and procedure for global transaction management is described in Figure 2. We review the basic protocol of OTM in the following subsection, and then compare two experimental results. More detailed can be found in the references [8, 9, 10].

5.1 Optimistic Ticket Method

The OTM uses *Ticket* to determine the relative serialization order of subtransactions in the local database systems. The *Ticket* is the regular data item, only one is required per LDBS. *Ticket* are processed by *Take-A-Ticket* operation which reads the *Ticket* value, increments it and writes is back using regular data operations. The *Ticket* value that is read by a subtransaction is later reported to the MDBS as logical timestamp of each site.

The global transaction is decomposed into subtransactions which are submitted to participating LDBS. When subtransaction enters the *prepared-to-commit* state, the MDBS takes the *Ticket* value of that site. If all subtransactions send *prepared-to-commit* messages and the MDBS takes all of their *Ticket* values, the OTM validates that the global transaction can be safely commit. If the local serialization orders are not compatible, the MDBS determines to abort. The validation is performed using *Global Serialization Graph*(GSG). The Edge(i,j) of GSG means *Ticket* operation of global subtransaction G_i is preceded by that of global subtransaction G_j at least one site. In other words, if *Ticket* value of G_i is smaller than G_j, edge from G_i to G_j is inserted to GSG.

Then the set of edges in GSG reflect the relative serialization order. If the

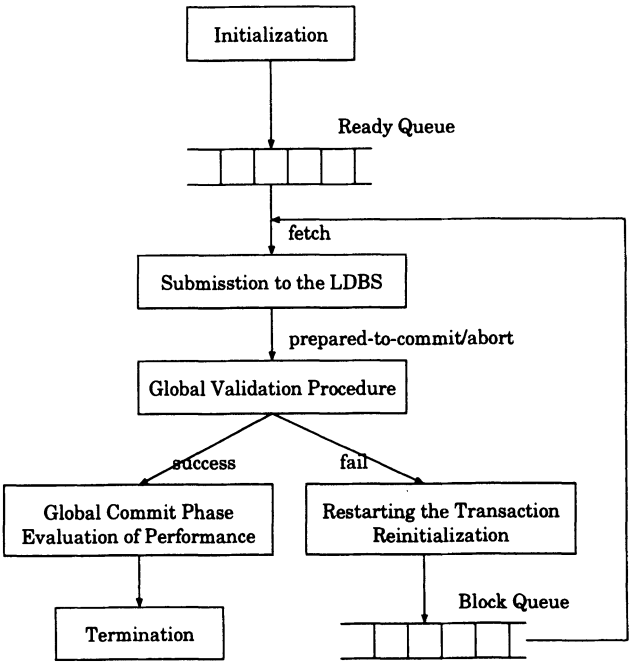

Figure 2 Overall Procedure of Global Transaction Management

GSG has a cycle, the MDBS sends abort message to all subtransactions of the global transaction and they must be restarted.

Therefore, although two global subtransactions of the same site access fully distinct data set, they should have relative serialization order caused by *Ticket* value. After the *Ticket* is taken by global subtransaction, any other global subtransaction cannot concurrently executed if the LDBS uses the concurrency control such a 2PL which is widely used in commercial DBMS. Hence, *Ticket* data causes the low degree of concurrency, while the global serializability is easily guaranteed by using it.

5.2 Experimental Results of Evaluation

The simulation testbed is implemented to evaluate the performance of proposed schemes and OTM in various configurations. The used modeling tool is *Simulation Language for Alternative Modeling* (SLAM II) [1] developed by Pristker & Associates. The major parameters names, its meanings and values of simulation experiments are described in Table 4

We mainly measure the number and ratio of restarting global transaction in validation procedure of GTM. The experimental results of proposed schemes and OTM are shown in following figures. We show the ratio of restarting global

Table 4 Values and Meanings of System Parameter

System Parameter	Meaning of Parameter	Value of Parameter
Max_Trans	Number of Maximum Global Transaction	[100, 900]
I_Total_Sites	Number of Local Sites	[10, 70]
I_DB_Size	Size of Local Database	30 Tables
OP_Time	Operation Time including Disk I/O	30 ms
Arrival	Mean Time between Inter Arrival	[2, 5]sec.
Ratio_Read	Ratio of Read Operation	[0, 100]percent
Ratio_R_Global	Ratio of Read Operation for Global Data	[0, 100]percent

transactions in Figure 3. The *x-axis* denotes the number of global transactions and *y-axis* denotes the ratio of restarting global transactions. As shown in Figure 3, the ratio of restarting global transaction increases as number of global transaction is large. The ratio of restarting global transactions in proposed method is smaller than that of OTM.

Figure 4 illustrates that the ratio of restarting global transactions decreases as the percentage of read operation in a global transaction is high. If the read operation percentage in a global transaction is high, the direct or indirect conflict between transactions is rapidly reduced. As expected, the ratio of restarting global transactions drops from the same value as OTM to 2 percent. However, the OTM is not affected by the ratio of read operation, because the global transaction makes a serialization order with all other global transactions in that site regardless of type of operations. As illustrated in previous subsection, even though a global transaction is *read-only* transaction, it has the explicit coercive serialization order with other *read-only* or*update* global transactions in that site. Hence, the ratio of restarting global transaction does not change by the percentage of read operation.

Particularly, in Figure 5, we measure the ratio of restarting global transactions corresponding to the portion of the read operation for global data. The x-axis represents the fraction of read operation for global data. The rest percentage is read operation for local data. In this experiment, we assume that the percentage of read operation in a global transaction is 50 percent. Therefore, if the percentage of read operation for global data is 50 percent, the *read operation for global data* of entire operations in a transaction is 25

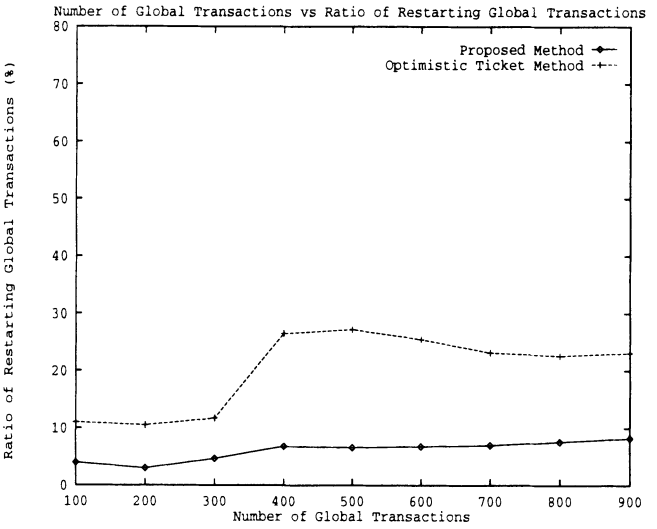

Figure 3 Restarting Global Transactions VS Number of Global Transactions

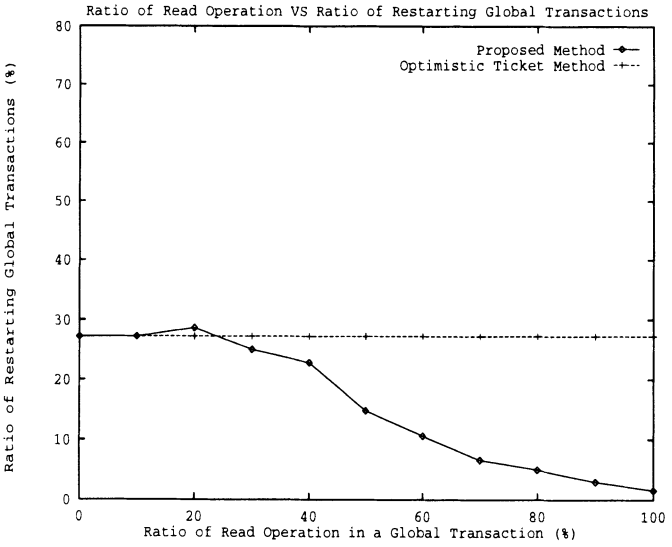

Figure 4 Restarting Global Transactions VS Read Portion of Global Transaction

percent. The rest 25 percent is the read operation for local data item and the half of entire operations are write operation. As shown in Figure 5, the ratio of restarting global transactions reduces as the percentage of read operation for global data increases.

Hence, the global transaction has more opportunities that can be safely committed as the transaction has more read operations as shown in Figures 4, 5. The fact that the more read operation, the more concurrency degree is widely proved in traditional research [2]. However, this fact is not adopted to the MDBS, because the MDBS prevent the global transactions from constructing the indirect conflict between themselves. Since the OTM proposes one of solutions by enforcing the explicit serialization order between all global transactions in a site, the ratio of read operation cannot affect the overall performance.

Figure 6 shows the ratio of restarting global transaction. The x and y axes represent the ratio of restarting global transactions and number of sites, respectively. If number of sites is more, the site contention of global transaction is reduced. The restarting global transaction decrease in both schemes since the site and data contention is reduced.

Finally, we observe that the ratio of read operation in a global transaction affects the overall performance of MDBS in proposed schemes, while does not in OTM.

6 CONCLUSION

Global Integrity Constraints arise naturally whenever the data that is semantically related is stored in different local database systems. The global integrity constraint is to specify the global configuration of the data that are considered semantically correct. In order to avoid global inconsistency, the enforcement of global integrity constraints must be observed and GTM has the responsibility for preserving the global consistency. Hence, in this paper, we propose the transaction model for preserving the global consistency and transaction management method based on our transaction model.

The basic problem of maintaining the global serializability of MDBSs is based on the indirect conflict between the global transactions, caused by the local transaction. Furthermore, global inter-site constraints must be preserved by the global transaction manager. Because of local autonomy of each LDBMS, the MDBS is unaware of indirect conflicts caused by certain local transaction and LDBS is also unaware of global inter-site constraints. Several solutions haven been proposed in many literature. However, most of them provide the low degree of concurrency or needed to use the some relaxed correctness criteria for the LDBMS. In this paper, we propose the appropriate transaction model that is applicable to the MDBS, and also propose the efficient optimistic concurrency control for the global transaction manager in MDBS, which guarantees the global serializability by preventing from the in-

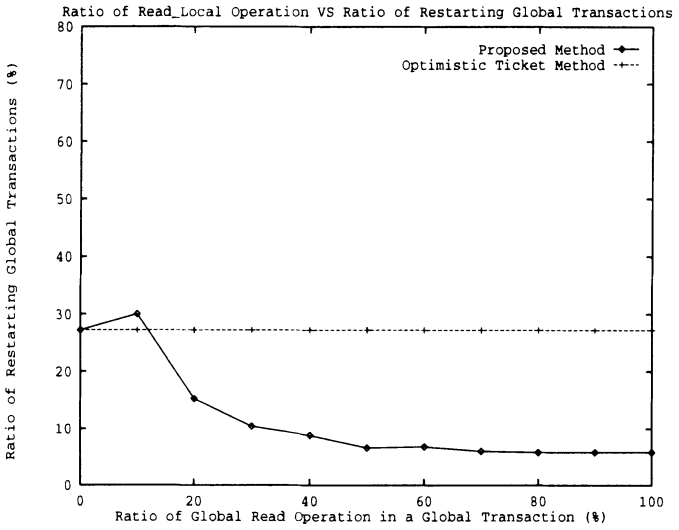

Figure 5 Restarting Global Transactions VS Local Read Portion of Global Transaction

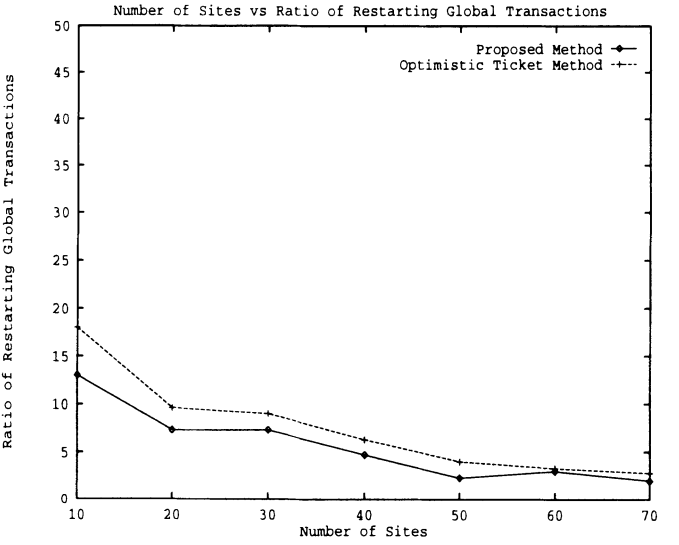

Figure 6 Restarting Global Transactions VS Number of Global Transactions

direct conflict between global transactions. We also prove the correctness of our proposed protocol. We are now working on implementing our proposed optimistic concurrency control protocol for MDBS.

REFERENCES

[1] A. Alan and B. Pritsker. *" Introduction to Simulation and SLAM II".* A Halsted Press Book, John Wiley Sons, 1986.

[2] Philip A. Bernstein, Vassos Hadzilacos, and Nathan Goodman. *" Concurrency Control and Recovery in Database Systems".* Addison-Wesley Publishing Company, 1987.

[3] Yuri Breitbart, Avi Silberschatz, and Glenn R. Thompson. "reliable transaction management in a multidatabase system". In *Proc. of the 1990 ACM SIGMOD International Conference on Management of Data*, pages 215–224, 1990.

[4] Stefano Ceri and Jennifer Widom. "managing semantic heterogeneity with production rules and persistent queues ". In *Proc. of the 19th International Conference on Very Large Data Bases*, pages 108–119, 1993.

[5] Sudarshan S. Chawathe, Hector Garcia-Molina, and Jennifer Widon. "a toolkit for constraint management in heterogeneous information systems". In *Proc. of the International Conference on Data Engineering*, pages 56–65, 1996.

[6] Weimin Du and Ahmed K. Elmagarmid. "quasi serializability : a correctness criterion for global concurrency control in interbase". In *Proc. of the 15th International Conference on Very Large Data Bases*, pages 347–355, 1989.

[7] Weimin Du, Ahmed K. Elmagarmid, and Won Kim. "maintaining quasi serializability in multidatabase systems ". In *Proceedings of the Research Issues in Data Engineering*, 1991.

[8] Dimitrios Georgakopoulos. "transaction management in multidatabase systems ". In *Ph.D. Thesis, University of Houston, Dept. of Computer Science*, 1990.

[9] Dimitrios Georgakopoulos, Marek Rusinkiewicz, and Amit Sheth. "on serializability of multidatabase transactions through forced local conflicts". In *Proc. of the 7th International Conference on Data Engineering*, pages 314–323, 1991.

[10] Dimitrios Georgakopoulos, Marek Rusinkiwicz, and Amit P. Sheth. "using tickets to enforce the serializability of multidatabase transactions". *IEEE Transactions on Knowledge and Data Engineering*, 6(1), 1993.

[11] Ashish Gupta and Jennifer Widom. "local verification of global integrity constraints in distributed databases". In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 49–58, 1993.

[12] Ahmed K. Elmargarmidand Jin Jing and Won Kim. "global commit-

ment in multidatabase systems ". In *Technical Report CSD-TR-91-107, Purdue university*, 1991.

[13] Ahmed K. Elmargarmidand Jin Jing, Won Kim, Omran Bukhres, and Aidong Zhang. "global commitability in multidatabase systems ". *IEEE Transactions on Knowledge and Data Engineering*, 8(5), 1996.

[14] Sharad Mehrotra, Rajeev Rastogi, Henry F. Korth, and Abraham Silberschatz. "non-serializable execution in heterogeneous distributed database systems". In *Proc. of the 2nd International Conference on Parallel and Distributed Information Systems*, pages 245–252, 1991.

[15] Sharad Mehrotra, Rajeev Rastogi, Henry F. Korth, and Avi Silberschatz. "relaxing serializability in multidatabase systems ". In *Proceedings of the Research Issues in Data Engineering*, pages 205–212, 1992.

[16] James G. Mullen, Won Kim, and Jamshid Sharif-Askary. "on the impossibility of atomic commitment in multidatabase systems". In *Technical Report SERC-TR-113, Purdue University*, 1993.

[17] Marek Rusinkiewicz, Amit Sheth, and George Karabatis. "specifying interdatabase dependencies in a multidatabase environment". *IEEE Computer*, 24(8):46–53, August 1991.

[18] Aidong Zhang and Ahmed K. Elmargarmid. "a theory of global concurrency control in multidatabase systems". In *Technical Report CSD-TR-92-098, Purdue University*, 1992.

Biography

Kyuwoong Lee(qlee@dblab.sogang.ac.kr) is a Ph.D. candidate in Dept. of Computer Science and a research fellow at Applied Science Research Institute at the Sogang University, Seoul, KOREA. His current research interests include concurrency control for real-time database system, distributed database system, and transaction scheduling for secure DBMS.

Seog Park(spark@dblab.sogang.ac.kr) is a professor of Dept of Computer Science of Sogang University, Seoul, KOREA. His research is focused on the Multi-Level Secure DBMS, Digital Library, Metadata Management for Data Warehousing, and Real-Time DBMS. He received a BS at Seoul National University, a MS and Ph.D. at Korea Advanced Institute of Science and Technology, Seoul, KOREA.