GLOBAL SERIALIZABILITY AND LOCAL AUTONOMY IN MULTIDATABASE SYSTEMS

Buhyun Hwang
Department of Computer Science
Chonnam National University
300, Yongbong-dong, Bukgu
Kwangju 500-757, Republic of Korea

Songchun Moon
Computer Engineering Division
Department of Information and Communication Engineering, KAIST
207-43, Cheongnyang-ni, Dongdaemun
Seoul 130-012, Republic of Korea

Global serializability and local autonomy in the MDBS (multidatabase system) are important issues since MDBS has been studied for the past five years. This paper proposes the transaction processing model for an MDBS and its three-level scheduling algorithm for global schedulers, for integrated local and global transaction schedulers (ILGSs), and for local schedulers to achieve global serializability, local autonomy, and higher concurrency. The most difficult problem to ensure global serializability for an MDBS is how to schedule global transactions so that their execution order and their serialization order at each site are identical. In our algorithm, this problem is resolved by two different ILGS modules: the first ILGS module is used in the case that local concurrency control (LCC) algorithm produces a history in which the serialization point of a transaction is the same as its serializable order, and the second ILGS module is used for each LCC algorithm that generates a serializable history. To achieve a high degree of concurrency, transactions are scheduled at the ILGS module, regardless of whether they are local or global, in a way that unnecessary abort or delay is not caused. By simply adding the ILGS level on top of each local DBMS, local DBMSs need not be changed at all. Therefore, local autonomy is guaranteed. Our algorithm also provides freedom from a deadlock at the global level and may be used even if data items are replicated at many sites.

1. INTRODUCTION

A multidatabase system (MDBS) is a distributed database system in which a number of different local database systems (LDBSs) are interconnected to facilitate the accesses of data across local database systems [Moon91]. The local database management systems (LDBMSs) may differ in data model, data definition and manipulation languages, transaction management policies, and internal data structures [Glig86]. The important characteristic of an MDBS is local autonomy, which includes design autonomy, commitment autonomy, and communication autonomy [Moon90].

We are concerned about the transaction management of an MDBS in this paper. What the transaction manager of MDBS pursues is to guarantee global database consistency while ensuring local autonomy and attaining higher concurrency or performance.

In MDBS, the traditional approaches to integrate heterogeneous concurrency control algorithms can be classified into two groups: bottom-up approach and top-down approach. The bottom-up approach [Pu88, Elma87, Elma89, Sugi87, Geo91] collects local information from each LDBMS at the global level and thereafter checks global serializability. This approach is optimistic and requires the modifications of each LDBMS to integrate LDBMSs. This approach requires considerable time and cost overhead because we have to modify all LDBMSs. Due to the modification, the bottom-up approach completely destroys the local autonomy of each LDBMS. The top-down approach [Sem91, Alex91, De88, Brei87, Brei89, Sark50, Lo90, Du91] maintains a global serialization order at local sites which has been determined already at a higher global level. It is pessimistic. The top-down approach does not require LDBMSs to be modified at all, and in this respect the local autonomy is completely ensured.

1.1 The Problems Inherent in Top-down Approach

There are four kinds of conflict that can occur between two different transaction operations. Let o, and o2 be operations of two different transactions, Tj and Ti, respectively, in a history H, where o, precedes o2.

Definition 1.1: Direct conflict (o1->o2)
We say that o, directly conflicts with o2 in H if both access the same data item at least one of them is a write operation. If o, directly conflicts with o2 in H, we say that Tj directly conflicts with Ti, i.e., Tj -> T,.

Definition 1.2: Indirect conflict (o1'->o2)
We say that o, indirectly conflicts with o2 in H if there exist operations o, o2, ..., o, of other transactions such that o, directly conflicts with o, in H, o, directly conflicts with o2 in H, ..., and o, directly conflicts with o, in H. If o, indirectly conflicts with o2 in H, we say that Tj indirectly conflicts with Ti, i.e., Tj -> T,.

Definition 1.3: Conflict (o1 -> o2)
We say that o, conflicts with o2 in H if o, directly or indirectly conflicts with o2 in H. If o, conflicts with o2 in a history, we say that Tj conflicts with Ti, i.e., Tj -> T,.

Definition 1.4: Possible conflict (o1'-> o2)
Let nws(Tj) denote a set of data items accessed by a transaction Tj. If nws(Tj) \cap nws(Ti) \neq \emptyset, we say that there is a possibility of a direct conflict between Tj and Ti. If nws(Tj) \cap nws(Ti) \neq \emptyset, nws(Tj) \cap nws(Ti) \neq \emptyset, and nws(Tj) \cap nws(Ti) \neq \emptyset, we say that there is a possibility of an indirect conflict between Tj and Ti. If there is a possibility of a direct or indirect conflict between Tj and Ti, we say that there is a possibility of a conflict between Tj and Ti, i.e., Tj -> T,.
Several serializable orders of transactions may be obtained for a given history. Among them, the serializable order (SO) in which transactions are arranged in an actual execution order is defined to be a physical serialization order (PSO).

Example 1.0: A physical serialization order (PSO)
Let R and W stand for a read operation and a write operation, respectively. T stands for a local or global transaction. The subscript used for R, W, and T denotes a transaction identifier. Suppose that a history H at a site is as follows:

$H = (T_1, T_2, T_3, T_4)$

The possible serialization orders of transactions, $T_1$, $T_2$, and $T_3$, are all combinations of $T_1$, $T_2$, and $T_3$, since there is no conflict among them. That is, $SO(H) = T_1 T_2 T_3$, $T_1 T_3 T_2$, $T_2 T_1 T_3$, $T_2 T_3 T_1$, $T_3 T_1 T_2$, $T_3 T_2 T_1$, $T_2 T_3 T_1$, $T_3 T_2 T_1$. But the PSO of $H$ is a single serialization order, which is $T_1 T_2 T_3$. That is, $PSO(H) = T_1 T_2 T_3$.

Let $H_i$ denote a local history at site $i$. Let $PSO(H_i)$ denote a physical serialization order of $H_i$. Let $GCPSO(H_i)$ denote a projection of globally committed transactions in $H_i$, and it is obtained from $H_i$ by deleting all operations that do not belong to the committed global transactions (GTs) in $H_i$. Let $PSOC(H_i)$ denote a physical serialization order of GC($H_i$), and $GCPSO(H_i)$ denote a projection of globally committed transactions in PSO($H_i$). PSO($H_i$) is in fact the actual execution order (EO) of GTs at site $i$.

In top-down approach ([Moon90]), most global schedulers process GTs so that $\cup_i PSOC(H_i)$ is acyclic. Since the top-down approach schedules global transactions without information about local transactions (LTs), undesirable problems may be caused: (1) unnecessary delay of GTs (See Example 1.1.), and (2) unnecessary abort of GTs (See Example 1.2.), (3) interdependency among GTs caused by a disagreement between $PSOC(H_i)$ and GCPSO($H_i$) (See Example 1.3).

Definition 1.5: If two transactions have an effect on each other either directly or indirectly, we say that there is an interdependency between the two transactions.

The site graph [Brei87] or transaction graph [Brei89] may be used as a scheduling strategy by a global scheduler. If the scheduler finds a cycle in the site graph or transaction graph, it must abort or delay the GT that produces the cycle since it assumes that if the graph has a cycle then there is a possibility of an interdependency among GTs due to an indirect conflict among GTs via LTs at the local level.

Example 1.1: Unnecessary delay of GTs
Let sites $S_1$ and $S_2$ contain data items $[a,b]$ and $[c,d]$, respectively. Suppose that global transactions, $G_1$, and $G_2$, respectively, are as follows:

$G_1: W_{g_1}(a) W_{g_1}(c) R_{g_1}(b)$
$G_2: W_{g_2}(a) W_{g_2}(c) R_{g_2}(b)$

Case 1. The order of GTs is scheduled by the unit of a transaction like in a schedule $GS_1$.

After $G_1$ has been scheduled, if $G_2$ is to be scheduled, a cycle is formed in a site graph and a transaction graph. Thus, $G_2$ cannot be scheduled and must be delayed until $G_1$ is committed.

Example 1.2: Unnecessary abort of GTs
Suppose that $G_1$ and $G_2$ are identical to those of Example 1.1 and global transactions, $G_1$ and $G_2$, are as follows:

$G_1: W_{g_1}(a) R_{g_1}(b)$
$G_2: R_{g_2}(a) W_{g_2}(d)$

Case 1. The case that GTs are scheduled by the unit of a subtransaction:
Consider that a global schedule, $GS_1$, of $G_1$ and $G_2$ is as follows:

$GS_1: R_{g_1}(a) R_{g_1}(b) W_{g_2}(c) W_{g_2}(d)$

At $t_1$, when the global scheduler based on the transaction graph schedules $W_{g_2}(d)$, a cycle is created. Thus, $G_2$ must be aborted.

Case 2. The case that cascading abort is caused:
Consider a global schedule, $GS_2$, as follows:

$GS_2: W_{g_1}(a) R_{g_1}(b) W_{g_2}(c) R_{g_2}(d) W_{g_2}(d)$

At $t_1$, when $R_{g_1}(c)$ is to be scheduled, a global scheduler based on the transaction graph will find a cycle in this graph. Thus, $G_2$ must be aborted and the transactions that are dependent on $G_2$ must also be aborted. Hence, a cascading abort happens.

The schedules, $GS_2$ and $GS_3$, are possible if any indirect conflict among GTs via LTs is not produced. Algorithms proposed in [Brei87, Brei88, Brei89] may cause the unnecessary abort (See Example 1.2). Although algorithm proposed in [Pu87] adopts the bottom-up approach, it may cause unnecessary abort since the order of order-elements in an order-vector must be the same at all sites, where the order-element is the serialization order of each transaction that is executed at each LDBS.

Example 1.3: The discrepancy between $PSOC(H_i)$ and $GCPSO(H_i)$
Let sites $S_1$ and $S_2$ contain data items $[a,b]$ and $[c,d]$, respectively. Suppose that global transactions, $G_1$, and $G_2$, a global schedule of them, $GS_1$, and a local transaction, $L_1$, are as follows:

$G_1: W_{g_1}(a) W_{g_1}(c) R_{g_1}(b)$
$G_2: W_{g_2}(a) W_{g_2}(c) R_{g_2}(b)$

$GS_1: W_{g_1}(a) R_{g_1}(b) W_{g_2}(c) R_{g_2}(b)$

Let $H_1$ and $H_2$ be a history at $S_1$ and $S_2$, respectively.

$H_1: W_{h_1}(a) R_{h_1}(a)$
$H_2: W_{h_2}(a) R_{h_2}(b) W_{h_2}(d) R_{h_2}(c)$

Let $H_1$, $H_2$, and $H$ be a history at $S_1$, $S_2$, and $H$, respectively.

$H: W_{h_1}(a) R_{h_1}(a)$
$H_1: W_{h_2}(a) R_{h_2}(b) W_{h_2}(d) R_{h_2}(c)$

In $H_1$, $G_1 \rightarrow' G_2$ in $H_2$, since $L_1 \rightarrow' G_2$ and $G_2 \rightarrow' L_1$. In $H$, $G_1 \rightarrow' G_2$. But, in $H_1$ and $H_2$, $PSOC(H_1)$ and $PSOC(H_2)$ are both $G_1G_2$. Since $GCPSO(H_1)$ is $G_1G_2$ and $GCPSO(H_2)$ is $G_2G_1$, a global history $GH = \{H_1, H_2\}$ is nonserializable.

In the top-down approach, the indirect conflicts among uncommitted GTs via LTs cannot be considered at the global level. Some of the indirect conflicts may create the discrepancy between $PSOC(H_i)$ and $GCPSO(H_i)$. Since, in top-down approach, this discrepancy may change the global serializable order of GTs given at the global level, it may destroy the global serializability as in Example 1.3. Algorithms proposed in [Brei87, Brei88, Brei89, Brat88] may produce this discrepancy between $PSOC(H_i)$ and $GCPSO(H_i)$ in LCC algorithms. It is said that concurrency control algorithms are static.
if the serialization point of a transaction is the same as its serializable order [Leeg0].

1.2 Solution for The Problems

To resolve the problems inherent in the top-down approach while preserving global serializability and local autonomy, a three-level scheduling algorithm is proposed in this paper. The first level and the third level of the three-level scheduling algorithm are the same as the global level and the local level of conventional MDBSs, respectively.

The role of the second level, that is, ILGS level, is to schedule global or local transactions so that:
1. unnecessary abort of GTs is not caused,
2. unnecessary delay of GTs is not caused, and
3. the serializable order of GTs in a global schedule given at the global level are effectively preserved at the local level.

That is, the interdependency problem caused by the discrepancy between PSOGC(Hj) and GCPSO(Hj) is not produced.

Items (1) and (2) above are necessary to improve performance. Item (3) is necessary to guarantee global serializability.

In Section 2, a transaction processing model for solving the problems inherent in the top-down approach is proposed. In Section 3, the three-level scheduling algorithm and two different ILGS modules used in the algorithm proposed in this paper are described. It is proved, in Section 4, that the three-level scheduling algorithm using either of these ILGS modules always guarantees global serializability.

2. TRANSACTION PROCESSING MODEL

Transaction processing model consists of three major components: global module, ILGS module, and local module, as shown in Figure 2.1: these are described in turn in detail. There is no delaying of global transactions at the global level. The sending and delaying of global transactions are determined by each ILGS module. Local module does not distinguish between a local transaction and a global transaction.

A GT is executed through three-step scheduling, performed by a global scheduler, an ILGS scheduler, and a local scheduler. An LT is executed through two-step scheduling, performed by an ILGS scheduler and a local scheduler.

3. THREE-LEVEL SCHEDULING

Three-level scheduling of MDBS is achieved by three components: a global scheduler, an ILGS scheduler, and a local scheduler. The LDBSSs do not distinguish GTs from LTs. The local schedulers of LDBSSs may be heterogeneous and any one of them need not be modified in order to be integrated. Therefore, the local autonomy of LDBSS is fully maintained.

For the three-level scheduling algorithm, we have assumed that (1) all local concurrency control algorithms produce only serializable local histories, (2) subtransactions of a GT have no value dependency among them, and (3) LTs may write only non-replicated data items. Since most local concurrency control algorithms produce a serializable history, assumption (1) is reasonable. Our algorithm is designed for parallel processing of subtransactions of a GT and thus assumption (2) is required. When a local transaction writes replicated data items, global database consistency cannot be preserved if the local transaction is not changed into a global transaction. Therefore, we need assumption (3).

We only focus on the scheduling algorithm of the global scheduler and that of the ILGS scheduler, since the LCC algorithm of the local scheduler need not be modified at all.

3.1 Global Module

The global scheduler in the global module schedules GTs by the unit of a transaction. The global scheduler simply sends both each subtransaction of a GT and a set of data items, to which the subtransaction accesses, to its corresponding ILGS module. The sending order of each subtransaction of a GT and the arrival order of it at each site must be identical.

3.2 ILGS Module

This section describes the role of ILGS module and two kinds of ILGS module. The role of an ILGS module is to (1) translate a global operation into local operations, (2) schedule LTs and GTs so that there is no conflict among uncommitted GTs to guarantee global serializability, (3) return the result of transactions to local user or GTM, and (4) perform the commit
failures, and communication failures, occur. But the algorithm identical at all sites or not. Thus, a GT which violates this relative order or of completed GTs on a basis of their ticket value is proposed in [Geo91] checks, of the global level, whether the and GTs that do not produce an indirect conflict among uncommitted transactions sent to LTM is represented by an IG. The reason in an IG. This principle guarantees global serializability (See Definition 3.1). IG consists of CCGs since it is a graph.

Definition 3.1: ILGS Graph (IG)
ILGS graph is IG = (V, E), where V is a set of transactions (local or global) and E is a set of edges. An undirected edge is called "connected" if for every pair of vertices there exists a path between them. A subgraph of a graph G is G' = (V', E'), where V' is a subset of V and E' is a subset of edges of E which connect vertices in V'. A subgraph G' = (V', E') is a connected component of the undirected graph G = (V, E) if G' is connected and there exists no other subgraph G'' = (V'', E'') of G which is also connected and either V' ⊆ V'' or E' ⊆ E''. We say that the subgraph G' is a connected component graph (CCG).

ILGS graph (IG for short) that is maintained in ILGS module is defined as below (Definition 3.1). IG consists of CCGs since it is a graph.

ILGS module consists of an ILGS translator, an ILGS scheduler, a manager of completed and committed transactions (MCCT), IG, local queue (LQ), and global queue (GQ), as shown in Figure 3.1. We say that a global subtransaction has been completed at the local level when it has finished its work at a single site, and a GT has been committed at the global level when all of its subtransactions have been completed.

ILGS translator translates a global operation into local operations. ILGS scheduler processes LTs and GTs so that an indirect conflict or a direct conflict among uncommitted GTs is not produced by the use of information in IG. The scheduled transaction is inserted into IG. MCCT returns the result of completed local transactions to local user and the result of completed subtransactions to GTM, and plays the role of a subordinate in the commit protocol. ILGS protocol does not affect the execution of an indirect conflict among uncommitted GTs in the future even if it is deleted from IG. ILGS scheduler and MCCT communicate with each other via IG for preventing a conflict among uncommitted GTs. Only can they update information in IG. GQ stores GTs along with their read sets and write sets by the order of transactions received from the global scheduler. LQ stores the LTs that are not sent to LTM.

The relationship about the sharing of data items among transactions sent to LTM is represented by an IG. The reason that an IG has no directed edge is that the execution order of transactions cannot be considered at the ILGS level.

In our three-level scheduling algorithm, we adopt the principle that there must be no conflict among uncommitted GTs in an IG. This principle guarantees global serializability (See Theorem 4.1). At the ILGS level, our algorithm schedules LTs and GTs that do not produce a conflict among uncommitted GTs. Hence, the completed subtransactions are never aborted except the case that some failures, i.e., transaction failures, site failures, and communication failures occur. But the algorithm proposed in [Geo91] checks, at the global level, whether the relative order of completed GTs on a basis of their ticket value is identical at all sites or not. Thus, a GT which violates this relative order is aborted.

If each CCG of IG contains at most one GT, there is no path among GTs in IG since there is no path between two CCGs. The fact that two transactions in IG have no path between them means that they do not directly or indirectly share data items with each other. Therefore, if each CCG of IG contains at most one GT, there is no direct or indirect conflict among uncommitted GTs.

In this section, two kinds of ILGS module are proposed to avoid the direct or indirect conflict among uncommitted GTs and to prevent global deadlock. The first ILGS module is used in the case that a local concurrency control (LCC) algorithm in an MDBS is static. The scheduling algorithms using two-phase locking (2PL) protocol, timestamp ordering (TO) protocol, optimistic, and value date are called static [Leu90]. The second ILGS module is used regardless of whether an LCC algorithm used in an MDBS is static or not.

3.2.1 ILGS Module Used When an LCC Algorithm is Static
This section describes an ILGS module used in the case that an LCC algorithm is static. In this section, it is proved that this ILGS module prevents a conflict among uncommitted GTs, and it is proved that the three-level scheduler using this module guarantees global serializability. Two components in this ILGS module, i.e., ILGS scheduler and MCCT, cooperate with each other to prevent a conflict among uncommitted GTs. This cooperation is achieved by inserting scheduled transactions into IG and deleting committed transactions from IG.

The following abbreviations are used for convenience.
1. CCG(T): a component graph that will contain or contains T.
2. Num_GT(CCG): the number of GTs in CCG.
3. SP(T): the serialization point of a transaction T.
4. FRONT(Q): the front of a queue Q
5. NONFRONT(Q): the position that is not the front of a queue Q
6. REAR(Q): the rear of a queue Q

Figure 3.1 ILGS Module
3.2.1.1 ILGS Scheduler Used When an LCC Algorithm is Static

ILGS scheduler consists of an algorithm for scheduling GTs and an algorithm for scheduling LTs. It sends alternately a LT and a GT to its LTM.

In the following algorithm for scheduling GTs, the mark "selected from nonfront" is used to denote that a GT is selected from NONFRONT(GQ). That is, it means that a GT marked as "selected from nonfront" is sent to LTM earlier than GTs that precede it in GTQ. This can produce high throughput for global transactions by preventing GTs from being delayed unnecessarily, but can create a deadlock at the global level (See Example 3.1.). But, we can avoid the deadlock by having a GT marked as "selected from nonfront" be executed without sharing data items with other transactions.

Algorithm for Scheduling Transactions (AST, for short):
1. IF LO is not empty,
   THEN invokes Algorithm for Scheduling LTs.
2. IF GO is not empty,
   THEN invokes Algorithm for Scheduling GTs.
3. REPEAT steps 2 and 3 UNTIL GP = REAR(GQ).

Let LP and GP denote a pointer of LO and a pointer of GO, respectively. LP and GP are moved from the front to the rear of each queue. Let G denote a global transaction pointed by GP, and SGPG denote a set of global transactions that precede G in GO.

Algorithm for Scheduling GTs (ASGT, for short):
1. GP is set to the FRONT(GQ).
2. IF GP is the FRONT(GQ),
   THEN G pointed by GP is selected.
   ELSE IF nseq(G) < nseq(C1) = e, for any C1 in SGPG,
       THEN G is selected and it is marked as "selected from nonfront",
       ELSE goto step 4.
3. Let CCG(r) be a component graph that will contain G. 
   IF Num_GT(CCG(T)) = 2,
       THEN G must be delayed at GO,
       ELSE BEGIN
           G is sent to LTM and G is removed from GO,
           and EXIT
       END
4. GP is increased by one.
5. REPEAT steps 2 and 4 UNTIL GP = REAR(GQ).

Let OLP denote a pointer that has an initial value of LP when the algorithm for scheduling LTs is called. It is used to notify whether LP circulates its LO once or not. Let LQ_SIZE denote a size of LO.

Algorithm for Scheduling LTs (ASLT, for short):
1. OLP is set to LP
2. Let T be a transaction pointed by LP.
3. Let CCG(T) be a component graph that will contain T.
   IF Num_GT(CCG(T)) >= 2 or (Num_GT(CCG(T)) = 1 
   and a GT in CCG(T) has been marked as "selected from nonfront"),
   THEN T must be delayed at GO and 
   ++LP mode LQ_SIZE;
   ELSE BEGIN
       T is sent to LTM and T is removed from LO,
       LP := (LP + 1) mode LQ_SIZE, and EXIT,
   END
4. REPEAT steps 2 and 3 UNTIL LP = OLP.

In the step 3 of ASLT, if a GT of the CCG that will contain a candidate LT to be sent to LTM has been marked as "selected from nonfront", the candidate LT is not sent to LTM and it remains in its LO. Thus, a GT that is marked as "selected from nonfront" is executed without sharing data items with other transactions.

ILGS scheduler schedules a transaction T so that Num_GT(CCG(T)) <= 1. If a CCG of IG has two or more GTs, i.e., there is a path between two transactions in a set of uncommitted GTs in the CCG, the possibility of an indirect or direct conflict between the uncommitted GTs may be introduced and thus global serializability may not be ensured. At all sites, if there is no indirect or direct conflicts among uncommitted GTs in each IG, the interdependency among GTs is not caused (See Theorem 4.1.).

3.2.1.2 MCCT Used When an LCC Algorithm is Static

An MCCT receives the result of transactions from its LTM and the result of the global decision from GTM, i.e., a globally (commit or abort) of a GT. Thus, the MCCT knows that (1) a LT has been committed, (2) a subtransaction has been completed and (3) a subtransaction has been committed.

If there is only an ILGS scheduler, it is not guaranteed that there is no conflict among uncommitted GTs (See Theorem 3.1.). Thus, an algorithm to delete only committed transactions, that have no effect on the creation of an indirect conflict among uncommitted GTs in the future, from IG is required.

Algorithm for Removing Committed Transactions from IG (ARCT, for short):
Let T be a committed local transaction or a completed or committed subtransaction.
1. IF T is a local transaction,
   THEN IF Num_GT(CCG(T)) = 1,
       THEN T does not have to be removed from CCG(T) of its IG,
       ELSE T is removed from CCG(T) of its IG immediately.
   ELSE IF T is completed,
       THEN T is not removed from its IG
       ELSE T is deleted from the CCG(T) of each IG and
       all committed local transactions that are reachable from T are deleted from the CCG(T).

Example 3.1: The case that a deadlock is produced at the global level when a GT that is marked as "selected from nonfront" is executed with sharing data items with other transactions.
Let S1 and S2 contain a set of data items, (a,b,c) and (x,y), respectively. Consider that global transactions, G1, G2 and G3, and local transactions, L1, and L2, are as follows.
G1: Rg2(a) Wg1(x)
G2: Wg2(b) Rg1(x)
G3: Rg3(y) Wg1(a)
L1: Wl1(a) Rl1(y)
L2: Wl2(y) Rl1(x)
Suppose that each subtransaction of G1 has been scheduled by each ILGS scheduler, and G1 and G2 are sent in the order, G1, G2, to each ILGS module by the global scheduler. Since the global scheduler schedules GTs by the unit of a transaction and the order in which the global scheduler sends GTs is effectively preserved at each site, the state of GO and IG at each site is like in Figure 3.2-(a). R and F in Figure 3.2 denote the rear of GO and the front of IG, respectively.
G1 at S1 denotes a subtransaction of G1 that accesses site S1. Consider the following scenario: (1) G1 at S1 is sent to LTM, (2) G2 at S1 is not sent to LTM since nseq(G1 at S1) < nseq(G2 at S2) =
the case that an LCC algorithm is static is used to schedule the three-level scheduling algorithm using the ILGS module. Therefore, the conflicts among uncommitted GTs are not created.  

By the following Theorems 3.1 and 4.1, it is proved that the three-level scheduling algorithm using the ILGS module used in the case that an LCC algorithm is static guarantees global serializability.

Theorem 3.1: At the ILGS level, if the ILGS module used in the case that an LCC algorithm is static is used to schedule transactions and send them to LTM and it is used to remove committed transactions from IG, there is no conflict among uncommitted GTs.

Proof: This ILGS module consists of ILGS scheduler and MCCT.
3.2.2 ILGS Module for Static or Non-Static LCC Algorithm

This section describes an ILGS module that is used regardless of whether an LCC algorithm is static or not. In this section, it is proved that this ILGS module prevents a conflict among uncommitted GTs, and it is proved that the three-level scheduler using this module guarantees global serializability. Two components in this ILGS module, i.e., ILGS scheduler and MCCT, cooperate with each other to prevent a conflict among uncommitted GTs and the disagreement between the execution order of GTs and their serialization order.

To produce a history in which the execution order of GTs and their serialization order are identical, MCCT of this module deletes committed transactions from IG by the unit of CCG. That is, each CCG must be removed from its IG after all transactions in the CCG have been committed. To remove CCGs from an IG as in the commit order of GTs as possible, the CCG containing the committed GT is marked as "committed" to prevent GTs from being connected to the CCG. After all, all GTs in the marked CCG are committed and the CCG is removed from its IG. Also, if a LT of a CCG that is composed of only local transactions is committed, the CCG is marked as "committed" to prevent the CCG as in the commit order of the LT as possible.

3.2.2.1 ILGS Scheduler Used When an LCC Algorithm Is Static or Non-Static.

ILGS scheduler of this module consists of an algorithm for scheduling GTs and an algorithm for scheduling LTs. An algorithm for scheduling GTs and an algorithm for scheduling LTs are identical to ASGT and ASLT, respectively, except that in step 3 of each of them, a transaction to be scheduled, T, must be delayed if a CCG that will contain T has been marked as "committed" by following MCCT.

Algorithm for Scheduling GTs (ASGT for short):
- IF Num_GT(CCG(G)) \geq 2 in the step 3 of ASGT, has been marked as "committed".

Algorithm for Scheduling LTs (ASLT for short):
- IF Num_GT(CCG(T)) = 2 or (Num_GT(CCG(T)) = 1 and a GT in CCG(T) is marked as "selected from nonfront") in the step 3 of ASLT, has been replaced by "IF Num_GT(CCG(T)) = 2 or (Num_GT(CCG(T)) = 1 and a GT in CCG(T) is marked as "selected from nonfront") in the step 3 of ASLT, has been marked as "committed".

3.2.2.2 MCCT Used When an LCC Algorithm Is Static or Non-Static.

In MCCT, an algorithm for removing committed transactions from IG is as follows.

Algorithm for Removing Committed Transactions from IG (ARCIT for short):
Let T be a committed transaction.
- IF T is a GT, Num_GT(CCG(T)) = 0, and CCG(T) has not been marked as "committed", THEN CCG(T) is marked as "committed".
- IF T is a GT, THEN CCG(T) is marked as "committed".
- IF T is a LT, THEN CCG(T) has been committed, THEN CCG(T) (i.e., all transactions in the CCG(T)) are removed from its IG.

By Theorems 3.2 and 4.1, it is proved that three-level scheduling algorithm using ILGS module for a static or non-static algorithm also guarantees global serializability.

Theorem 3.2: At the ILGS level, the ILGS module for a static or non-static algorithm is used to send transactions to LTM and to remove committed transactions from its IG, there is no conflict among uncommitted GTs.

Proof: This ILGS module consists of ILGS scheduler and MCCT.
1. By ILGS scheduler, a transaction T is scheduled so that Num_GT(CCG(T)) \leq 1. Hence, each CCG of IG has at most one GT. Therefore, the conflict among uncommitted GTs is avoided.
2. For any CCG, and CCG, in each IG, rwsen(CCG, T) \subseteq rwsen(CCG, T). By MCCT, the committed transactions are removed by the unit of a CCG from each IG. Thus, no conflict among uncommitted GTs is created by the removed CCG.

4. GLOBAL SERIALIZABILITY IN THE THREE-LEVEL SCHEDULER

We have shown that the three-level scheduler using two kinds of ILGS module proposed in this paper never generates the direct or indirect conflicts among uncommitted GTs at each site (or GTs in each IG) by Theorems 3.1 and 3.2. In this section, it is proved that the three-level scheduling process guarantees global serializability if there is no direct or indirect conflict among the GTs in each IG of our three-level scheduler.

Theorem 4.1: Global serializability is guaranteed if there is no direct or indirect conflict among uncommitted GTs in each IG.

Proof: Suppose that there are a global transaction G to be scheduled, a scheduled but not committed global transaction G' and a set of G's, denoted by GTS. A site(G) denotes a set of sites that are accessed by G. Let IS be site(G) \cup site(G'). |IS| denotes the number of sites in IS. We check the global serializability for following three cases.

Case 1. |IS| = 0, for any G in GTS.
- G and G' are not related at all since there is no common site accessed by both G and G'. Hence, G and G' may be executed in any order.

Case 2. |IS| = 1, for some G's in GTS.
- There is one common site accessed by G and G'. In this case, if site(G) \cap site(G') = \emptyset, site(G) \cap site(G') = \{S\}, and site(G) \cap site(G') = \{S, S'\}, and site(G') \cap site(G) = \{S\}.
- The transaction graph is cyclic like in Figure 4.1-(a).
- There may be the indirect conflicts as follows: G \rightarrow G' at site S, G' \rightarrow G' at site S', and G' \rightarrow G' at site S'.
- Assume that there are above indirect conflicts. Let IG, IG', and IG' be an IG at sites S, S', and S", respectively. In our algorithm, only the GTs that are deleted from an IG may affect uncommitted and uncommitted GTs in the IG. For above conflicts to be created, the state of GTs at each site must be as follows:
  1. At site S, G has been removed from IG, G' must be in IG', and the state of G' must be uncommitted and uncommitted.
  2. At site S', G' has been removed from IG', G must be in IG', and the state of G must be uncommitted and uncommitted. Since G, G', and G' are all uncommitted, G and G' and G' and G, and G must be in IG, IG', and IG', respectively.
- Therefore, there is no indirect conflict at each site. This violates
the above assumption. Thus, above indirect conflicts are not created in our algorithm. Therefore, any interdependency among GTs is not produced.

Case 3. \(|I S| \geq 2\), for \(G\) in \(IG\).

There are two or more common sites accessed by both \(G\) and \(G\). Consider the case that \(|I S| = 2\). The transaction graph for this case contains a cycle like Figure 4.1-b. Assume that \(G \rightarrow G\) at site \(S_2\) and \(G \rightarrow G\) at site \(S_2\). To produce \(G \rightarrow G\), at site \(S_2\), \(G\) must be removed from \(IG\), and the state of \(G\) must be uncompleted and uncommitted. To produce \(G \rightarrow G\), at site \(S_2\), \(G\) must be removed from \(IG\), and the state of \(G\) must be uncompleted and uncommitted. Since \(G\) and \(G\) are both uncommitted, both \(G\) and \(G\) must be in \(IG\), and \(IG\). Therefore, this violates the above assumption since there is no indirect conflict between \(G\) and \(G\). Thus, above indirect conflicts are not created in our algorithm. Hence, in case \(|I S| = 2\), the interdependency between two GTs is not produced. In the case where \(|I S| > 2\), the same procedure with case of \(|I S| = 2\) is applied.

Above three cases are all cases that must be considered. Therefore, the global serializability is guaranteed if there is no direct or indirect conflict among uncommitted GTs in each IG.

5. CONCLUSIONS

In this paper, the model of transaction management in an MDBS and its three-level scheduling algorithm are proposed to resolve the problems inherent in the top-down approach. In this paper, two different ILGS modules are proposed on basis of whether LCC algorithms are static or not. Even if any one of these modules is used, it has been shown that our algorithm completely preserves global serializability and local autonomy.

If the ratio of LTs which do not create an indirect conflict among uncommitted GTs is very high, it is considered that our algorithm can obtain higher concurrency than the algorithms that unnecessarily abort or delay transactions.

If there is an interdependency among GTs, some global transactions that actually cause the interdependency must be aborted. The cost to abort the transactions is very high. Our algorithm prevents global transactions from being aborted due to the interdependency, i.e., the violation of global serializability, although there is some delaying of these transactions. This delaying or aborting is necessary for any type of algorithm. Since our algorithm does not delay or abort global transactions unnecessarily, we can obtain a high degree of concurrency.

The cost overhead involved in the ILGS level would be high. The processing time to send global and local transactions to LTM so that the interdependency is not created, and the storage space to keep the information needed for ILGS scheduler are the overhead.

In our proposed algorithm, ILGS scheduler must still be refined to reduce cost overhead and to increase performance. The performance analysis for our algorithm through simulation is also required to compare our algorithm with other algorithms.

REFERENCES


