Achieving high degree of concurrency in multidatabase transaction scheduling: MTOS

Kyungcheol Sohn *, Songchun Moon

Database System Research Laboratory, Graduate School of Management, KAIST, 207-43 Cheongryangri, Dong-Daemun Gu, Seoul 130-012, South Korea

Abstract

A multidatabase system (MDBS), which is also called a heterogeneous distributed database system (HDDBS) or federated database system (FDBS), is a facility that allows users or applications to access data located in multiple local database systems (LDBSs), each of which is autonomously operated. Several practical MDBS concurrency control schemes have been proposed whilst maintaining global consistency without compromising local autonomy. However, they could degrade either local concurrency of local transactions or global concurrency of global transactions for the purpose of ensuring the global serializability. In this paper, we propose a new ticket-based global concurrency control scheme that employs multiple tickets at each site. The basic idea is that, with proper consideration of the potential cyclic global serialization orders, subtransactions with the possibility of the local indirect conflicts must access the same ticket at their sites, otherwise they are allowed to access the different tickets. This scheme alleviates the blockage on local resources, and, as a result, increases both global concurrency and local concurrency, whilst preserving the local autonomy. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Multidatabase; Local autonomy; Concurrency control; Indirect conflict; Ticket

1. Introduction

A multidatabase system (MDBS) [1], which is also called a heterogeneous distributed database system (HDDBS) or federated database system (FDBS), is a facility that allows users or global applications to access data stored in multiple local database systems (LDBSs), each of which is autonomously operated. The objective of an MDBS is to integrate the various LDBSs to allow global updates. Two different types of transactions are executed in an MDBS environment: (1) local transactions and (2) global transactions each of which consists of a number of subtransactions.

The issue on transaction management in an MDBS is how to preserve the global consistency in the presence of local and global updates. To overcome the difficulty, an MDBS has to guarantee the global serializable execution of local and global transactions [5]. As a result of recent studies [2,6,8], the task of ensuring global serializability has reduced to either preventing or avoiding both the direct conflicts and the indirect conflicts. Direct conflicts could exist between two different subtransactions, whereas indirect conflicts could be caused if two global transactions do not directly conflict but conflict via some intermediate local transactions. Detecting direct conflict is not
difficult because subtransactions are executed under the MDBS control. Detecting indirect conflict, however, is difficult because local transactions are executed outside of the MDBS control and are invisible at the global level.

In the literature, the optimistic ticket method (OTM) [8] appears to be the first practical solution that does not violate the local autonomy and is applicable to all LDBSs that ensure local serializability. The basic idea in OTM is to ensure that the overhead incurred by the use of a single ticket for each site is considered to be severe. The ticket operation consists of two consecutive operations, reading the ticket value and writing ticket value with an incremented value. Thus, the serialization order of subtransactions at the each site is indicated by the ticket value that has been read, and, as a result, the ticket operations are used to guarantee the relative local serialization order of subtransactions at all local sites to be consistent with the ticket value obtained by subtransactions at their sites.

With a single ticket mechanism, nevertheless, subtransactions may be either blocked or aborted at the time for them to access a ticket due to its unavailability. Moreover, to prevent the possibility of local indirect conflicts, subtransactions are forced to take the ticket at their sites of origin in order to be serialized according to the order in which they take their tickets. Unfortunately, this way of chained conflict enforcements could bring unnecessary conflicts between two subtransactions that are in fact irrelevant to each other, even though they do not have potential local indirect conflicts. Example 1 shows this. Consider an MDBS consisting of three sites: S1, S2 and S3 (Fig. 1). Data items a and b are assumed to be stored at S1; x and y are stored at S2, and v and w are stored at S3. In Fig. 1, the ticket object at Si is denoted by ti. Let G1,k denote the subtransaction of a global transaction G1 at site Si. If G1,1 is denoted for instance as rG1,1(a)rG1,1(t1)wG1,1(t1 + 1), this means that G1,1 reads a, and then issues two consecutive ticket operations which consist of reading t1 and incrementing t1.

**Example 1 (Unnecessary conflicts due to single chained-conflict enforcement).** Consider Fig. 1. To simplify the discussion, let us assume that subtransactions take ticket in transaction-number order at each site in order to assign the same relative local serialization order. This implies that, for instance, G1,2 and G2,2 are executed concurrently, but G1,2 takes t2 and commits before G2,2 takes t2 in a serial fashion.

Note that in the local schedule H1 we have a forced local serialization order G1 → G3, since G1,1 takes t1 and commits, and then G3,1 takes t1 and commits. Similarly, we realize G1 → G2 via t2 from H2 and G2 → G3 via t3 from H3. Accordingly, the global serialization order G1 → G2 → G3 and G1 → G3, which is to be the union of H1, H2 and H3, is acyclic. Therefore, global serializability is preserved.

However, we observe that to prevent cyclic global order G1 → G2 → G3 → G1, only two sites among S1, S2 and S3 require forcing conflict on the tickets. For instance, we assume that S1 produces a forced local serialization order G1 → G1 via t1 and S2 produces G1 → G2 via t2. Note that the cyclic global serialization order will not occur regardless
of the serialization order between \( G_{2,3} \) and \( G_{3,3} \), which could be generated \( G_3 \rightarrow G_1 \) or \( G_3 \rightarrow G_2 \), since a possible global serialization order caused by \( G_{2,3} \) and \( G_{3,3} \) is either \( G_1 \rightarrow G_2 \rightarrow G_3 \rightarrow G_1 \) or \( G_1 \rightarrow G_2 \leftarrow G_3 \leftarrow G_1 \). Hence, \( G_{2,3} \) and \( G_{3,3} \) do not need to be forced to conflict with regard to \( t_3 \).

Note that, in Fig. 1, if we do not use ticket operations, global serializability could not be maintained, since \( L_1 \) could incur a local indirect conflict between \( G_1 \) and \( G_3 \) in the form of \( G_3 \rightarrow L_1 \rightarrow G_1 \). This implies that, if the local serialization orders in \( S_2 \) and \( S_3 \) are assumed to be \( G_1 \rightarrow G_2 \) and \( G_2 \rightarrow G_3 \), respectively, \( H_1, H_2 \) and \( H_3 \) altogether could produce a non-serializable global order \( G_1 \rightarrow G_2 \rightarrow G_3 \rightarrow G_1 \).

In this paper, we propose a new MDBS concurrency control scheme to overcome the concurrency degradation in the previous schemes. Basically, this scheme employs multiple tickets at each site in order to reduce the unnecessary conflicts due to the single chained-conflict enforcement. At each site, then, we handle the multiple tickets based on the ticket-allocation mechanism that could make a global subtransaction incur a conflict on a ticket among its multiple tickets, with the possibility of a cyclic global path in mind. Based on this philosophy, in [11,12], we have proposed the basic ideas and the preliminary version of this paper. Our scheme ensures the global serializability in that the subtransactions of each global transaction are synchronized to have the same relative serialization order at all sites.

The rest of the paper is organized as follows. The next section reviews existing MDBS concurrency control schemes, and then examines their concurrency degradation for the global concurrency of global transactions and the local concurrency of local transactions. In Section 3, we describe an MDBS model, including underlying assumptions, on which our proposed scheme will work. In order to overcome the concurrency degradation in existing schemes, Section 4 proposes our MDBS concurrency control scheme, called a multiple-ticket ordering scheme (MTOS), to handle multiple tickets at each site whilst preserving the global serializability. In Section 5, we prove its correctness and compare its performance with other existing schemes. Finally, conclusions and future extensions appear in Section 6.

2. Related works and problems

Users of database system access the database by invoking transactions. A transaction \( T_i \) is a sequence \( \text{read}(r_{T_i}) \) and \( \text{write}(w_{T_i}) \) started by \( \text{begin}(b_{T_i}) \) and terminated by \( \text{commit}(c_{T_i}) \) or \( \text{abort}(a_{T_i}) \). The local schedule in a site is serializable if and only if its local serialization graph is acyclic [3]. A union of local serialization graphs is called a global serialization graph. A global schedule is globally serializable if and only if its global serialization graph is acyclic [5]. Unless otherwise noted, we refer to serializability as conflict serializability as defined in [3].

Many techniques for ensuring global serializability in an MDBS have been addressed in the literature. Among them, several proposals [2,4,5,8,10,13,14] could be applicable to MDBS environments whilst maintaining the global serializability without modifying the concurrency control mechanism in LDBSs. From the viewpoint of how to deal with the local indirect conflicts, these proposed schemes could fall into one of the two different approaches.

(1) Avoiding possibility of local indirect conflicts

[2,5,10]. This Avoiding-Conflict approach avoids the indirect conflicts by controlling the submission order of global transactions under the control of the global transaction manager (GTM), whilst preserving the local autonomy. For the purpose of it, GTM requires some information about the local transactions in LDBSs or imposes a restriction on the concurrency control method in LDBSs.

The site-graph scheme [5] avoids possibility of indirect conflicts under the observation as follows: whenever global transactions execute at the same sites, there is a possibility of indirect conflicts between them due to the local transactions. This scheme could correctly detect the possible indirect conflicts between global transactions. In addition, it handles these indirect conflicts by the submission delay of the subtransaction. This scheme, however, could cause some drawbacks. The degree of the global concurrency goes down, since no global
transactions can be executed in the same two LDBSs concurrently. Example 2 illustrates this drawback.

**Example 2 (Low degree of global concurrency due to execution policy in site-graph scheme).** Consider an MDBS consisting of three sites: $S_1$, $S_2$ and $S_3$ (Fig. 2). Let us assume that global transactions are submitted in transaction-number order at each site to simplify the discussion. Initially, $G_1$ is submitted and executed since its edges do not create a cycle in the site graph. Note that, when $G_2$ starts, its edges create a cycle, so $G_2$'s submission should be delayed due to the execution policy of the site-graph scheme. Similarly, we realize that $G_3$'s submission is delayed due to a cycle, since it blocks the execution of other global transactions until it is completed. Thus, the concurrency degradation for global transactions is considered to be severe, even if this scheme could not affect the local concurrency of local transactions.

In the rigorous scheduler scheme [2], if all the LDBSs produce the rigorous schedules, the global serializability is maintained by controlling the commitment (execution) order of subtransactions. However, this scheme could not be applicable if all the concurrency control mechanisms in LDBSs do not satisfy this restriction. It could also degrade global concurrency and local concurrency, since both local transactions and global subtransactions could be executed in a serial fashion. Note that a rigorous scheduler prevents write–read, write–write, and write–write conflicts between uncommitted transactions.

In serialization event scheme [10], it utilizes information on serialization events contained in concurrency control mechanism of LDBSs. A serialization event of a transaction is a distinguished action that determines the serialization order of the transaction in a schedule. For instance, in the two-phase locking (2PL) protocol, the action obtaining the last lock could be defined as the serialization events. However, this scheme could degrade global concurrency and local concurrency, since it preserves the global serializability by controlling the serialization events of subtransactions in a serial fashion.

(2) **Forcing conflicts between subtransactions** [4,8,13,14]. This Forcing-Conflict approach prevents indirect conflicts by forcing the serialization order between subtransactions under the control of local transaction managers (LTMs), whilst preserving the local autonomy. For this purpose, each LTM requires meaningless operations in a global subtransaction or incurs each global subtransaction to access the ticket at its LDBS with creation of a ticket in advance. In addition, Forcing-Conflict does not require any restrictions or information on LDBSs other than only serializable and recoverable [3] schedules whilst ensuring global serializability.

OTM [8,9] uses a ticket to force conflicts between subtransactions at each LDBS in order to prevent indirect conflicts. However, as shown in Example 1, OTM leads to the unnecessary conflicts. Furthermore, this unnecessary conflict could incur the concurrency degradation on not only the global concurrency of global transactions, but also the local concurrency of local transactions. Example 3 illustrates this.

**Example 3 (Low degree of global concurrency and local concurrency due to unnecessary conflicts in OTM).** Recall Example 1 in which there are six tickets altogether: $t_{1,1}, t_{2,1}$ at $S_1$, $t_{1,2}, t_{2,2}$ at $S_2$ and $t_{1,3}, t_{2,3}$ at $S_3$ (Fig. 3). In Example 1, even if subtransactions have been executed concurrently until their ticket operations, their executions will be
blocked due to the ticket availability. For instance, in Fig. 1, \( G_{1,2} \) takes \( t_2 \) and commits, and thereafter \( G_{2,2} \) is allowed to take \( t_2 \) and commits in a serial fashion due to ticket conflicts. Notice that this sort of execution results in a strict serialness.

However, in Fig. 3, \( G_{1,2} \) and \( G_{2,2} \) could be executed and committed concurrently with the ticket conflicts on \( t_{1,2} \) and \( t_{2,2} \), respectively. Nevertheless, they never violate the global serializability since a possible global serialization order caused by \( G_{1,2} \) and \( G_{2,2} \) is either \( G_1 \rightarrow G_2 \rightarrow G_3 \rightarrow G_1 \) or \( G_1 \leftarrow G_2 \rightarrow G_3 \leftarrow G_1 \). Note that the forced conflict \( G_1 \rightarrow G_2 \) is definitely unnecessary one. Thus, the unnecessary conflicts could degrade the global concurrency.

Unfortunately, this unnecessary conflict could also degrade the local concurrency of the local transactions which wait the resources held by global transactions due to possible blockage of a ticket. At \( S_1 \), if \( L_1 \) wait the data item \( b \) held by \( G_{3,1} \), for instance, \( L_1 \) could cause cascading wait for \( L_2, L_3, \ldots, L_n \), where \( L_i \) (\( 2 \leq i \leq n \)) is a set of local transactions and \( L_j \) wait the resources held by \( L_{j-1} \). Thus, the degree of the local concurrency could be degraded drastically, since the waiting time of local transactions due to the cascading wait is increased.

Multidatabase timestamp mechanism (MTSM) [4] is a decentralized deadlock-free refinement of the OTM. MTSM, using timestamp instead of timeout in OTM, assigns global serialization order to global transactions. Thus, the tickets they should read are determined in advance. As a result, MTSM assures that global transactions are serialized in the timestamp order at all LDBSs. Similarly, the drawbacks identified in OTM exist.

There is another sort of approach to enforce conflict by using the dummy read [13] and the extra operation [14]. These schemes require only the insertion of retrieval operations into subtransactions. Even if read operation causes less blocking than update operations, they have some problems. For example, when subtransactions diversely access different kinds of data, they may lead long appendices of read operations. Also, at any time, at most one subtransaction can be active at any site. Thus, these methods could also degrade the global concurrency and the local concurrency, and in addition suffer wasteful resource consumption.

A key feature of the MDBS is each LDBS and its application must continue to operate without any modification in the MDBS environment. The integrated LDBSs may belong to distinct, and possibly competing business organizations (e.g., competing computerized reservation agencies). It is undesirable, hence, to use a concurrency control scheme where a site belonging to a competing organization can block the resources in LDBSs, degrade local concurrency, and degrade global concurrency for the purpose of ensuring the global serializability; a phenomenon that can occur under the previous schemes. Consequently, we need an alternative to the previous schemes that alleviates the blockage on local resources, and also increases global concurrency, whilst preserving the local autonomy.

3. The MDBS model

A multidatabase system (MDBS) consists of a global transaction manager (GTM) and a number of local transaction managers (LTM), one for each local database system (LDBS) located at a site \( S_i \) (Fig. 4). Transactions in an MDBS are of two types: (1) A local transaction (\( L \)) is a transaction that accesses only one LDBS. (2) A global transaction (\( G \)) is a transaction that accesses more than one LDBS. A global transaction \( G_k \) is composed of a set of global subtransactions, \( \{G_{k,1}, G_{k,2}, \ldots, G_{k,n}\} \), where the subtransaction \( G_{k,i} \) accesses LDBS at a site \( S_i \). Basically, the MDBS processes each
global transaction via interactions between GTM and LTMs. Global transactions are managed by GTM. Upon receiving a global transaction, GTM decomposes it into several subtransactions that are submitted to LTMs individually. Each LTM is responsible for submitting the operations of global subtransactions to the associated LDBS and interacting with GTM to achieve global serializability. Once a subtransaction is submitted to an LDBS, it is treated the same way as if it were a local transaction.

As a necessary assumption of global serializability for the algorithm being discussed in the next section, we assume that each global transaction can have at most one subtransaction at each LDBS. It is necessary since global serializability may not be preserved if a global transaction has more than one subtransaction at a given LDBS [7]. We also assume that each LDBS provides a visible prepared-to-commit state for its transactions, and also has the ability to resolve local deadlocks and recover from local failure. This assumption is made to focus our study on the concurrency control aspects. Finally, we assume that each LDBS could provide information on either the maximum execution time among its local transactions or the transaction identifiers for the local transactions which are being run at it. To simplify the discussion, we assume that each LDBS provides the local transaction identifiers for its LTM. This assumption is necessary for LTMs to prevent an indirect conflict among irrelevant global transactions, each of which use different tickets for each other, at the point of a subtransaction submission in advance. However, no restriction is imposed on the concurrency control mechanisms in LDBSs except for requiring only local serializability.

4. Multiple-ticket ordering scheme

In this section, we present our MDBS concurrency control scheme, called a multiple-ticket ordering scheme (MTOS). We first describe the background notion employed by our scheme. We then introduce algorithms for global concurrency control under MTOS.

4.1. The notion of multiple-ticket ordering

To preserve global serializability in an MDBS, it is necessary to handle the local indirect conflicts between subtransactions. In order to handle the local indirect conflicts, in Forcing-Conflict approach the conflict enforcements (i.e., ticket conflict, dummy read, and extra operation) causes all subtransactions to form a single conflict chain in an arbitrary order, for instance, in the form of \( G_1 \rightarrow G_2 \rightarrow \cdots \rightarrow G_n \), even if some subtransactions could not be included in the single conflict chain. The following lemma is proven in [8,9,13,14].

**Lemma 1** (Effect of single conflict chain). *If the conflict enforcements make all the subtransactions incur a single conflict chain, the local indirect conflicts are not allowed between subtransactions.*

The impact of a single conflict chain on the degree of local concurrency is considered to be severe, since subtransactions could incur a long-duration block, as in Example 3, on the local resources which are waited by local transactions. Our goal in developing MTOS, therefore, is to reduce the possible blockages of local resources in a single conflict chain. For this purpose, we split a single conflict chain into a multiple conflict chain, called a *multiple ticket ordering* (MTO), by means of...
of multiple tickets at each site. With the consideration of local autonomy, the number of tickets at each site could be defined statically at each LDBS’s pleasure. In order to split a single conflict chain, we detect an unnecessary conflict enforcement in the chain with cycle detection by checking a form of graph, which is a bipartite graph whose node set is formed by the set of global transactions and sites whilst edges connect a transaction with all sites where its subtransactions execute. In MTOS, it is called a cycle detection graph (CDG) which is similar to the site graph [5]. The following lemma is proven in [5].

**Lemma 2** (Possibility of non-serializable schedule). *If a bipartite graph between global transactions and sites contains a cycle, there is a possibility of violation of global serializability due to a local indirect conflict.*

Hence, this method of checking a graph successfully could detect the groups of global transactions, each of which incurs its own cycle. For instance, consider four global transactions: $G_1$ is to run at $S_1$ and $S_2$, $G_2$ is to run at $S_2$ and $S_3$, $G_3$ is to run at $S_1$ and $S_2$, and $G_4$ is to run at $S_2$ and $S_3$. If GTM receives them in transaction-number order, there could be two irrelevant cycle groups: one is $G_1 - S_1 - G_3 - S_2 - G_1$, the other $G_2 - S_2 - G_4 - S_3 - G_2$.

According to the result of graph checking, MTO could be composed of a number of single conflict chains. Hence, to preserve global serializability, MTOS should handle the local indirect conflicts between subtransactions of global transactions in the same relevant group, and at the same time between subtransactions of global transactions in irrelevant groups. The indirect conflicts in a relevant group are prevented by conflict enforcements via tickets in such a way of forming a single conflict chain. This sort of a conflict chain is formed in the way that the subtransactions of the two adjacent global transactions in a cycle path access the same ticket at their participatory sites. The indirect conflicts between subtransactions in the irrelevant groups are forbidden by the subtransaction delay, which is identical to that of the site graph [5]. Note that, the indirect conflicts could be avoided in case only one global subtransaction is submitted during the execution life of a local transaction. Hence, if a subtransaction unfortunately happens to be involved in an indirect conflict, its submission could be delayed by its LTM until the local transaction that could incur an indirect conflict is committed. The following lemma formalizes the characteristics of an indirect conflict between two subtransactions.

**Lemma 3** (Origin of local indirect conflict). *If only one global subtransaction is submitted during the execution life of a local transaction, the local indirect conflicts are not allowed between subtransactions.*

**Proof.** At a site $S_i$, suppose that $G_{a,k}$ and $G_{b,k}$ are the subtransactions of global transactions $G_a$ and $G_b$, respectively, and also $L_1$ is a local transaction that shares resources with $G_{a,k}$ and $G_{b,k}$. Also, suppose that $G_{a,k}$ is submitted before $G_{b,k}$. If a local scheduler produces a local schedule $H_k = L_1G_{a,k} \cdots G_{b,k}L_1$, there could exist a local indirect conflict between $G_{a,k}$ and $G_{b,k}$ in a form of $G_{b,k} \rightarrow L_1 \rightarrow G_{a,k}$, where ‘$L_1 \rightarrow G_{a,k}$’ is formed by $L_1G_{a,k}$ and ‘$G_{b,k} \rightarrow L_1$’ by $G_{b,k}L_1$. This indirect conflict in $H_k$ could be prevented, if the submission of $G_{b,k}$ is delayed until $L_1$ is committed. Consequently, an indirect conflict is sufficiently prevented in a way that only one global subtransaction is submitted during the execution life of a local transaction. □

**Lemma 4** (Prevention of local indirect conflict). *The local indirect conflicts are not allowed between subtransactions if one of the following conditions is satisfied: (1) Two global subtransactions are directly in conflict. (2) Only one global subtransaction is executed during the execution life of a local transaction.*

**Proof.** The first condition is proved by Lemma 1, since direct conflict creates the conflict chain between them. Hence, for instance, $H_k$ in Lemma 3 will be not allowed by the local scheduler, since the direct conflict between $G_{a,k}$ and $G_{b,k}$ causes locally non-serializable schedule, $G_{a,k} \rightarrow G_{b,k}$ and $G_{b,k} \rightarrow L_1 \rightarrow G_{a,k}$. The second condition is proved
as follows. The indirect conflict in Lemma 3 could be prevented, if the submission of $G_{i,k}$ is delayed until $L_i$ is committed. Hence, an indirect conflict is sufficiently prevented in the way that only one global subtransaction is submitted during the execution life of a local transaction.

4.2. Cycle detection graph

To examine a possibility of unserializable global schedules in cycle detection phase, we employ a cycle detection graph (CDG), which is similar to the site graph [5]. The difference is that the site graph concerns only to interleave global transactions excluding indirect conflicts, whilst, in addition to the site graph, CDG determines what tickets the subtransactions should be enforced in order to prevent an indirect conflict whilst delaying the submission of it. In the commitment validation phase, GTM imposes a compatible serialization order between global transaction at all LDBSs to preserve global serializability.

In the cycle detection phase (Fig. 5), when a new global transaction $G_i$ is submitted to the LDBS, the edge $(G_i, G_k, \emptyset)$ without the ticket number is entered into CDG for each site $S_k$ at which $G_i$ will run. GTM then examines a possibility of global cycles incurred by $G_i$ in CDG. Notice that a message between GTM and LTM is denoted as $msg(G_{i,k}, \{t_{j,k}, \ldots, t_{j,k}\})$, where $t_{j,k}(0 \leq m \leq s)$ is a usable ticket number to enforce

4.3. Multiple ticket ordering algorithms

In order to implement MTOS, our scheme employs multiple tickets at each site in order to create the multiple chained-conflict enforcement. It then maintains them through three phases: the cycle detection phase, the cycle resolution phase, and the commitment validation phase. In the cycle detection phase, GTM examines a possibility of non-serializable global schedules incurred by a new global transaction. In the cycle resolution phase, each LTM forces a subtransaction to conflict on one among its own multiple tickets in order to prevent an indirect conflict whilst delaying the submission of it. In the commitment validation phase, GTM imposes a compatible serialization order between global transaction at all LDBSs to preserve global serializability.
conflict at Sk. If CDG(Gi) has a cycle, there is a possibility of non-serializable global schedules. In order to prevent these cyclic paths, hence, subtr(Gi) is forced conflict on the same tickets which are used by subtransactions of adjacent(Gi) to be involved in the same conflict chain. Consequently, GTM sends subtr(Gi) along with ticket numbers used by adjacent(Gi) to LTMs at site(Gi). Otherwise, GTM sends subtr(Gi) without the ticket-allocation information to LTMs at site(Gi).

In the cycle resolution phase (Fig. 6), on a basis of messages from GTM, if LTM receives only the subtransaction of Gi, it makes that subtransaction of Gi incur a conflict on an available ticket among its own multiple tickets with the ticket availability in mind. By available we mean that LTM is allowed a free choice of a ticket among its own multiple tickets under its judgement on ticket availability. On the contrary, if LTM receives the subtransaction of Gi with the transaction numbers for global transactions adjacent to Gi, the subtransaction of Gi is enforced a conflict on the same ticket which at the moment is being used by the global transactions adjacent to Gi, with proper consideration of availability for its tickets. At the point of subtransaction submission, in addition, LTM has to delay the submission of the received subtransactions only if it is involved in an indirect conflict, which could be occurred in subtransactions between irrelevant groups. Hence, LTM has to compare the current transaction identifiers about the local transactions with previous transaction identifiers, which was obtained at the point of the previous received subtransaction, for the purpose that only one global subtransaction is allowed to be submitted during the execution life of a local transaction.

In the commitment validation phase (Fig. 7), we employ a transaction commit graph (TCG), which is similar to the global serialization graph (GSG) [8], to preserve the compatible serialization orders among global transactions. The TCG can be described as follows:

**Definition 2 (Transaction commit graph).** A transaction commit graph (TCG) is a directed graph $T; E$ whose set of nodes $T$ consists of a set of global transactions. An edge $Gi \rightarrow Gj$ is in $E$ if both $Gi$ and $Gj$ take the same ticket at $Sk$, and ticket value of $Gi$ is a smaller than that of $Gj$.

The validation test will allow global transactions to commit only if their relative serialization order is the same in all participating sites. TCG keeps all committed global transactions to validate a global transaction in the prepared-to-commit state. If subtransactions of Gi complete all of their operations, and enter their prepared-to-commit state, then GTM validates Gi with TCG.

```
procedure Cycle Resolution
input: msg(Gi: possible ticket numbers) from GTM;
output: msg(Gi: assigned ticket numbers) to GTM;
begin
Wait msg(Gi); /* s = the number of usable tickets */
when (Gi.s)
  Select an available ticket $t_0$ among multiple tickets:
    assigned_ticket_no := $t_0$;
  when s ≥ 2
    Select $t_k \in \{t_0, t_1, \ldots, t_{s-1}\}$ by the availability of tickets:
    assigned_ticket_no := $t_k$;
end case
if (Gi.k means a indirect conflict) then
  Delay submission of Gi;
end if
Execute Gi with the conflict on a ticket of assigned_ticket_no;
Send msg(Gi: assigned ticket numbers) to GTM;
end Cycle Resolution;
```

```
procedure Commitment Validation
input: receive prepared_to_commit from all LTMs of Gi;
output: send validation result to all participatory LTMs;
begin
Receive prepared_to_commit from site(Gi);
for (each Gi ∈ G) do
  for $Sk \in \{site(G_i) \cap site(G_k)\}$ do
    if (both Gi and Gk have taken the same ticket $t_{out}$) then
      if (value($t_{out}$.Gi) < value($t_{out}$.Gk)) then
        Add a directed edge < Gi, Gk > to TCG;
      else
        Add a directed edge < Gk, Gi > to TCG;
      end if
    end if
  end for
end for
if (the augmented TCG contains a cycle) then
  Send abort_subtransaction for subtr(Gi) to site(Gi);
  Remove all incident edges of Gi in TCG;
  Send commit_subtransaction for subtr(Gi) to site(Gi);
end if
end Commitment Validation;
```

Fig. 6. Cycle resolution phase of GTM.

Fig. 7. Commitment validation phase of GTM.
first attempts to insert edges between $G_i$ node and nodes corresponding to every recently committed global transactions $G$ that have taken the same ticket as $G_i$. If $G_i$ takes $value(t_{m,k})$ at $S_k$, an edge $G \rightarrow G_i$ is added to TCG, where $G$ is a set of global transactions that access the same $t_{m,k}$ but obtains a smaller $value(t_{m,k})$ than that of $G_i$. Otherwise, if $G_i$ takes $value(t_{m,k})$ at $S_k$, an edge $G_i \rightarrow G$ is added to TCG, where $G$ is a set of global transactions that access the same $t_{m,k}$ but obtains a larger $value(t_{m,k})$ than that of $G_i$. Thereafter, if all such edges can be added without creating a cycle in TCG, $G_i$ is validated, and thus committed. Otherwise, $G_i$ does not pass validation. Hence, its node together with all incident edges is removed from the graph. Consequently, $G_i$ is aborted, and thereafter restarted.

In summary, our scheme processes a global transaction $G_i$ as follows. Initially, when a new global transaction $G_i$ enters, GTM sets a timeout for $G_i$ to prevent a deadlock, and also makes use of CDG to detect potential cyclic global serialization orders created by $G_i$. Then, GTM submits subtransactions of $G_i$ with the ticket-allocation information to their participatory LTMs. At each site, thereafter, on a basis of the ticket-allocation information LTM assigns a ticket among its multiple tickets to a subtransaction, and then executes it. Finally, if all subtransactions of $G_i$ complete a task and also enter their prepared-to-commit state, then GTM validates $G_i$. The validation test will allow global transactions to commit only if their relative serialization order is the same in all participating sites.

5. Correctness and comparisons

In this section, we first show that MTOS algorithms are correct in terms of global concurrency control in MDBS environments. We then compare our scheme with the previous schemes from the viewpoint of global concurrency and local concurrency.

5.1. Correctness of MTOS algorithms

The correctness criteria for our scheme is the global serializability [3], which is the most commonly used an MDBS environment. MTOS preserves global serializability if (1) the participating LDBSs generate serializable schedule, (2) a global transaction has at most one subtransaction at each LDBS, (3) each LDBS provides a visible prepared-to-commit state. We now describe a theorem that specifies correctness of MTOS.

**Theorem 1.** MTOS guarantees the global serializability.

**Proof.** To prove the correctness of MTOS we will show that:
1. CDG detects a potential cyclic path between global transactions.
2. MTO algorithms maintain multiple conflict chains without local indirect conflicts.
3. TCG maintains the same relative serialization order of a global transaction at all participating sites.

The first claim is proved as follows. By Lemma 2, CDG correctly detects potential cyclic global serialization orders due to local indirect conflicts induced by local transactions. We now prove the second claim. The conflict enforcement between adjacent subtransactions in a single conflict chain occurs via the same ticket at a site. In addition, each chain in multiple conflict chains uses the different tickets among multiple tickets at a site. By Lemma 4, hence, the indirect conflicts in a single conflict chain and between multiple conflict chains are prevented by MTO algorithms. Finally, we prove the third claim. The order in which subtransactions take their tickets reflects their relative serialization order. Thus, the subtransactions of global transaction $G_i$ have the same relative order between them if TCG ($G_i$) is not involved in a cycle.

5.2. Comparison to other schemes

Among the previous schemes, site graph (SG) scheme in Avoiding-Conflict approach and OTM scheme in Forcing-Conflict approach ensure the global serializability, and also require only local serializable schedule whilst preserving local autonomy. Hence, we compare MTOS with these representative schemes. Three schemes have de-
sirable property that they maintain global serializability without modifying LDBSs whilst preserving local autonomy. However, from the point of view of the degree of global concurrency and the degree of local concurrency, some differences are recognized between them.

First, consider the degree of global concurrency. In case of SG, no global transactions could be executed in the same two LDBSs concurrently. Thus the degree of concurrency is considered to be rather low. In case of OTM, all sites have only single ticket. Accordingly, even if they have been executed concurrently until their ticket operations, their executions may be blocked due to the competition to take a single ticket. Consequently, each subtransaction cannot but obtain a single ticket and commit in a serial fashion. In case of MTOS, each site may have multiple tickets. With proper consideration of local indirect conflict, it handles tickets to reduce subtransaction blocking time due to the ticket availability. MTOS, thus, provides more degree of concurrency than OTM, since the duration of subtransaction blocking is reduced by avoiding the unnecessary conflicts.

Second, consider the degree of the local concurrency. In case of SG, it does not consume the resources in LDBSs to prevent local indirect conflict, because the global serializability is preserved from a GTM point of view by means of only the cycle checking of the site graph. Thus, SG could not incur the concurrency degradation of local concurrency due to its control scheme. In case of OTM, there could be unnecessary conflict in a global serialization order. Thus, it could degrade the degree of concurrency not only for global transactions but also for local transactions which wait resources held by global transactions. In case of MTOS, a global serialization order is divided by multiple tickets. Thus, as compared with OTM, the degree of the local concurrency could be increased, since the waiting time of local transactions due to the cascading wait is decreased.

Finally, as compared to OTM, our scheme ensures global consistency while maintaining local autonomy at the price of multiple tickets. The major overhead of having multiple tickets comes from the delay of subtransaction submission in cycle resolution phase. An overhead with respect to the delay of subtransaction submission could occur for the purpose of preventing local indirect conflicts between subtransactions in the irrelevant groups. This sort of delay is inevitable if a subtransaction unfortunately happens to be involved in an indirect conflict. Accordingly, if a subtransaction is involved in indirect conflicts, it is delayed until a local transaction that causes indirect conflicts is committed.

However, this overhead is considered to be not significant because it will be occurred only for some subtransactions, each of which is involved in indirect conflicts, and also the delayed subtransactions do not holding the resources in LDBSs. Moreover, to minimize the submission delay in LTM, the intrinsic characteristics of each subtransaction can be used. For example, if two subtransactions in the irrelevant groups conflict directly at some LDBS, the delay between them can be minimized by controlling the order in which they issue their conflicting operations. Further research is in progress on these investigations.

6. Conclusions

In this paper, we proposed a new ticket-based mechanism, MTOS, for scheduling global transactions in an MDBS. Our scheme can contribute to the MDBS performance at the price of multiple tickets with respect to the degree of global concurrency and the degree of local concurrency. More specifically, the global concurrency degree can be enhanced due mainly to the reduction of both the duration of subtransaction blocking and the frequency of subtransaction aborting, which are caused by the unnecessary conflicts. The local concurrency degree also can be improved due mainly to the decrease of the waiting time of local transactions, which are waiting resources held by the stand-by global transactions on a hot-spot ticket.

There are two approaches according to the action taken by the MDBS concurrency control [1]: the one is optimistic, the other pessimistic. A pessimistic approach does not generate global transaction aborts by means of global transaction delay but may result in low concurrency, while an opti-
mistic approach may increase concurrency but may result in a large number of transactions aborts. In this regard, MTOS is an optimistic concurrency control scheme since the global serializability is ensured via validation at committing time, but can also be used in a pessimistic way. In this case, global transactions are assigned a priori a global serialization order, and the tickets they should read are determined in advance. If a transaction submits its operation outside of a local site ticket order, it waits.

References


Kyungcheol Sohn is a Ph.D. student at the graduate school of management in Korea advanced institute of science and technology (KAIST). He received his B.S. degree in computer science from the Dongguk University, Korea, in 1985, and M.S. degree in computer science from KAIST, in 1987. He has been working for Korea Securities Computer Corporation (KOSCOM) since 1987. His research interests include multidatabase system, mobile database system, and transaction management.

Songchun Moon earned his Ph.D. in Computer Science from the University of Illinois at Urbana-Champaign in 1985. He has been working for KAIST since then. He was a distinguished scholar at the Hungarian Academy of Science and still serves for EUROMICRO as director since 1991. He has developed IM, a multi-user relational database management system which is the first prototype ever in Korea in 1990, and also, DIIME, a distributed database management system in 1992, which is another first prototype ever in Korea. His current interests are in database security management and also in enterprise-wise data modeling. NON – STOP, such a data modeling methodology widely adopted by a number of Korean industry players such as Samsung, has been invented and reported by him. NON-STOP Data Modeling has been authored by him in 1997.