A Top-Down Approach For Two Level Serializability

M. OUZZANI
Institut d'Informatique - USTHB
El Alia B.P. No. 32 Bab Ezzouar
ALGER ALGERIE
Fax: (213-2) 75-82-01

M.A.ATROUN
Institut d'Informatique - USTHB
El Alia B.P. No. 32 Bab Ezzouar
ALGER ALGERIE
Fax: (213-2) 75-82-01

N.L.BELKHODJA
Institut d'Informatique - USTHB
El Alia B.P. No. 32 Bab Ezzouar
ALGER ALGERIE
Fax: (213-2) 75-82-01

Abstract

Concurrency control has received considerable attention in multidatabase systems because of their characteristics such as heterogeneity and autonomy. Particulary, various concurrency control protocols have been developped in the litterature. In this paper, we present a protocol that guarantees the two level serializability criterion and built up according to the top-down approach.

1. Introduction

A **multidatabase** is a collection of **pre-existing databases**. Each database is controlled by a particular **local DBMS (LDBMS)** that is **autonomous** and eventually **distributed**. The system permitting the logic integration of this DBMS is called multidatabase system (**MDBS**). The LDBMSs are heterogeneous in the sense that they can use different data models, query languages, transaction management strategies, ...etc. The local autonomy of the LDBMSs constitutes the principle characteristic of these environments, it can be viewed under different aspects [SL90]: Design autonomy, execution autonomy, communication autonomy and association autonomy.

Two types of transactions are present in the multidatabase, the **local transactions** derived from pre-existing local applications, and **global transactions** derived from the new global applications that span data located in several LDBMSs. Hence, the **global transaction manager (GTM)** should guarantee the correct execution of global transactions, even in the presence of local transactions of which the GTM is not aware. This task becomes complicated because of the autonomy requirements of LDBMSs. These LDBMSs are not willing to co-operate with the GTM for correction of global executions.

The transaction management in multidatabases is considered as **hierarchical** [ED90]. At each site there is a local DBMS that manages transactions executed in the local site. On top of these DBMSs, there is a global DBMS (or GTM) that manages global transactions accessing more than one LDBMS

**Proceedings of the 20th VLDB Conference
Santiago, Chile, 1994**

and that "co-ordinates" local executions to ensure correct execution of global transactions.

An execution of global transactions is considered as correct with respect to a correctness criterion. The GTM should synchronise these transactions so that this criterion is ensured. Several criteria have been proposed in multidatabase environments, they derived all from the classical criterion of **serializability**. These various criteria can be distinguished by the level of correction they permit and by the restrictions they impose. The first criterion used is the **global serializability**. This criterion is [GM91, MRKS91, DE89] difficult to maintain and proposed algorithms result in a poor performances. It is, hence necessary to adopt a weaker correctness criterion. To achieve this, we have first the notion of **quasi serializability** [DEK91, DE89]. Based on the integrity constraints, another weaker criterion has been proposed in [MRKS91], the **two level serializability**. This criterion defines a greater set of correct schedules than the first two criteria and has also the advantages of being simple, allowing a high degree of concurrency, and not violating the local autonomy.

By its hierarchical nature, the concurrency control in multidatabases can be characterised by two different approaches: the **top-down** approach and the **bottom-up** approach. The former has better advantages for multidatabases. In this paper, we propose a new protocol of concurrency control that maintains the two level serializability with respect to a top-down approach.

The remainder of the paper is organised as follows. In section 2, we define a model for multidatabases and some concepts related to these environments. A two level serializability correctness criterion is described in section 3. Section 4 treats the global concurrency control and the top-down approach. We describe in section 5 our new protocol. In section 6, we compare the protocol with another proposed in [MRKS91]. We conclude by enumerating perspectives for our work.

2. Preliminary Definitions

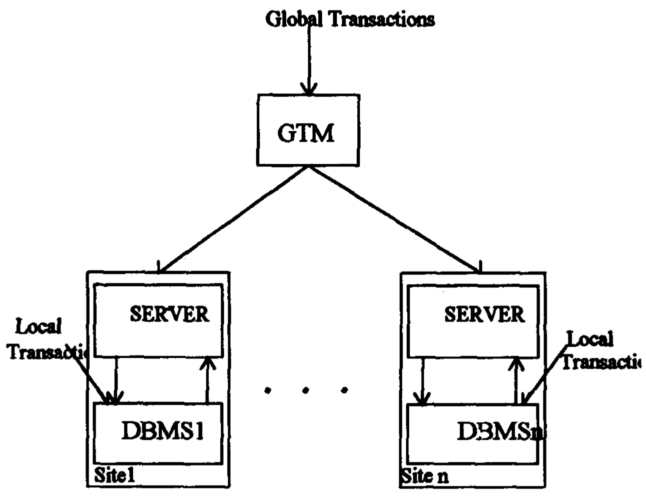

Fig 2.1 The MDBS Model

Two types of transaction are supported in the multidatabase:

- **Local transactions**, which access data managed by only a single DBMS. These transactions are executed by the LDBMS, outside of MDBS control.
- **Global transactions**, which are those executed under MDBS control. A global transaction consist of a number of subtransactions, each of which is an ordinary local transaction from the point of view of the LDBMS where it is executed.

The GTM is the part of MDBS responsible for global transactions management. The various servers constitute the interface MDBS/LDBMS. Global subtransactions are submitted by GTM to these servers and then to the LDBMS. We have then two level of control, the GTM and the LDBMSs, this leads to the definition of two types of schedules.

Definition 2.1:

A local schedule at site s_k, denoted S_k, is a sequence of local and global transactions operations resulting from their execution at site s_k. ❑

Definition 2.2:

A global schedule S is a partial ordered set of all operations belonging to local and global transactions such that, for any local site s_k, a projection of S on the set of global and local transactions executing at site s_k is the local schedule S_k. In fact, $S=\{S_1, S_2, ..., S_n\}$. ❑

In general, two types of integrity constraints may be present in the multidatabase:

• **Global integrity constraints**: they may span more than one site.

• **Local integrity constraints**: they can involve only items at a single site.

We can also partition the data of a LDBMS in two layers:

• **Global data**: the set of data involved in a global integrity constraint.

• **Local data**: the set of data not involved in a global integrity constraint.

It is clear that the local transactions may not modify global data, otherwise the global consistency may be not ensured, since the pre-existing local transactions are not aware about these new global integrity constraints.

3. Two Level Serializability

A correctness criterion precise the conditions under which a concurrent execution of transactions is considered correct. The level of correction can change from one criterion to another, but in general, it should preserve the database consistency and guarantees that the transactions "see" consistent data. Consistency is defined by a set of integrity constraints that links data items. The correction concerns, in multidatabases, the global schedule but also the local schedules. The global serializability criterion has been first used [GRS91 ,BS88, AGMS87, Sug87]. The inherent problems of this criterion have lead to new criterion that are adapted for multidatabases. These criterion derive all from the notion of serializability. They can be distinguished by the level of correction and also by the imposed restrictions. First, we have the notion of quasi-serializability [DEK91, DE89]. Later on, another criterion based on integrity constraints have been proposed. Their objective is to guarantee that accepted global schedules are strongly correct.

Definition 3.1 [MRKS92]:
An execution is strongly correct if the final state produced is consistent and the state read by each transaction is consistent. ❑

In multidatabases, two extremes cases can be considered:
i) No global integrity constraints. The local DBMSs are independent.
ii) There exist some integrity constraints that link data located in different LDBMSs.

In the former case, it suffice to ensure that a schedule is strongly correct if all local schedules are serializable. We obtain then the **local serializability** criterion.

Definition 3.2 [BGMS92]:
A global schedule S is locally serializable **(LSR)** if for every site s_i, the local schedule is serializable. ❑

However this criterion is applicable only if [BGMS92] the transactions (global and local) are **Local Database Preserving (LDP)**. A transaction is LDP if it preserves consistency of a given site regardless of the state of other sites.

For the second case, it is presented in [MRBKS91] the **Two Level Serializability** criterion. It is defined as follows.

Definition 3.3 [MRKS91]:
A global schedule S is Two Level Serializable **(2LSR)** if all local schedules are serializable and the projection of S on global transactions is serializable. ❑

This criterion has been designed to reach the two following requirements, 1) it should be easily implementable, and 2) schedules satisfying the criterion must preserve database consistency. The preservation of consistency depend on the nature of transactions and integrity constraints. The ease of implementation is provided by the fact that the control can be made over global transactions regardless of local ones. Nevertheless, the strong correction is guaranteed only under some conditions that depend on permitted accesses for global and local transactions.

In [BGMS92], the authors require that transactions are LDP and **Global Database Preserving (GDP)**. A transaction is GDP if it preserves global constraints regardless of the state of local data items. It may be possible to relax these requirements in some particular cases.

To simplify the work required to guarantee the 2LSR, we can state a theorem for 2LSR similarly to those for global serializability [BS88].

Theorem 3.1:
Let S be a global Schedule. S is 2LSR, iff:

i) the local schedules are serializable, and

ii) the projections of local schedules on the global subtransactions define the same relative serialization order for global transactions. ❑

The concurrency controller must ensure that the global transactions are serialized, among them solely, in the same order in all sites. We see that the 2LSR criterion offers us a more flexible control than global serilizability. In this paper, we develop a new protocol of global concurrency control for 2LSR that adopt a top-down approach.

4. The Top-Down Approach For Concurrency Control

Concurrency control is an activity that co-ordinates concurrently executed transactions so that they interfere with each other in a correct manner. The autonomy property complicates this task in multidatabase environments. Indeed, the synchronisation of global transactions, that are also under control of various autonomous LDBMSs, cannot be done in co-ordination with this LDBMSs. The concurrency control in multidatabases is hierarchical, we can classify the adopted strategies into two categories: bottom-up and top-down.

In bottom-up approach, the various LDBMSs schedule independently the global transactions. It is the GTM responsibility to detect and resolve the incompatibilities among local orders. This approach suffers from high rate of abortion of global transactions, due to incompatibilities among local executions. In contrary, in the top-down approach, the GTM determines a global serialization order of global transactions before submitting them to local sites. This order is then enforced at the local sites with, possibly, different techniques. The comparison of the two approaches [ED90] points out the advantages of top-down approach over bottom-up one. The advantages are: free global deadlock, no abortion and weak load of communication. However, the bottom-up approach is best in terms of concurrency.

A detailed study of top-down approach and its applications for global serializability and quasi serializability are presented in [ED90]. We resume here the principal aspects of this study. There are two basic steps in a top-down approach:

1) determining an order (O) of global transactions at global level, and

2) enforcing this order at local level.

The problem is with enforcing an order at the LDBMSs. The solution of this problem depends on the concurrency control strategies LDBMSs used and also on the autonomy requirements of LDBMSs.

Enforcing a global pre-specified order can be achieved by two different ways:

•Enforcing the order by controlling the submission of global transactions.

•Enforcing the order by controlling the execution of global transactions.

The second technique is applicable only if the LDBMS relax their autonomy requirements. The figure 4.1 depicts a protocol that controls the submission of global subtransactions at local level by use of a server process. $G_i(O)$ is the set of global subtransactions that are submitted with a pre-specified order O.

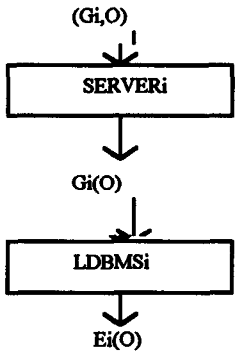

Fig 4.1 Enforcing a pre-specified order O by a server process.

The server process can be considered as a function that has global subtransactions and a pre-specified order as inputs and that delivers a submission policy as output. This policy is constituted of two components:

1) The order of global subtransactions submission

2) The conditions under which each global subtransaction is submitted. This conditions depend on the correction criterion and the local concurrency controllers.

5. The Protocol

The study of top-down approach and its useful advantages for global concurrency control in multidatabases lead us to design a new protocol of concurrency control that adopts this approach.To ensure the 2LSR, it only needs to guarantee that the projection of a global schedule on the global transactions is serializable. To do so, we must ensure that the projections of all schedules on global subtransactions are compatible. The pre-specified order is a serialization order relative to only global transactions. We assume complete autonomy of LDBMSs, the pre-specified order is then enforced, if necessary, by controlling the submission of global transactions. We determine now the condition under which a global subtransaction is submitted. This condition depends on two aspects:

1) The knowledge of serialization orders of a global subtransactions in a local schedule.

2) When is necessary to enforce a serialization order among two global subtransaction.

5.1. Knowledge Of Serialization Orders

This knowledge depends closely on types of local schedules. We consider then the case when this type is known and the contrary case.

5.1.1. Unknown Local Schedules

We assume only that the local schedules are serializable. It is then possible to use the idea of ticket developed in [GRS91]. The ticket is a particular object in a LDBMS that gives the serialization order of each global transaction that update it. In our case, we use the fact that the execution order of conflicting operations over a given object is compatible with their serialization order. Therefore, at each site, a particular object "SR" is inserted. The object "SR" is, of course, undere the LDBMS controls. This object is updated every time a global subtransaction accesses the site, and an acknowledgement is send to the server. The submission strategy in this case is then:

Order: The global subtransactions are submitted to the LDBMS in an order compatible with the pre-specified order. ❑

Condition1: A global subtransaction will not be submitted to the LDBMS until all previous ones have updated the object "SR". ❑

5.1.2. Particular Local Schedules

In this case, we assume that the type of local schedules is known. We describe, in subsequent, the submission strategies for two types of schedules that are useful in multidatabase systems.

a) Serialization Points And Dependencies

A serialization point (or serialization event) is a particular action that determines the serialization order of a transaction in a schedule. In 2PL method [BHG87], the action of obtaining the last lock can be considered as a serialization point. For TO [BHG], this particular action corresponds to timestamping.

Definition 5.1 [BGMS92]:
Let S be a serializable schedule consisting of transactions $\{T_1,T_2,...,T_n\}$. We say that schedule S is a serialization point-schedule if and only if there exist a mapping sp from transactions to actions such that:

1. $sp(T_i) = o_k$ where $o_k \in T_i$; and
2. If $sp(T_i)$occurs before $sp(T_j)$ in S, then there exist a serial schedule equivalent to S in which T_i precedes T_j. ❑

The GTM should know for each LDBMS, the action that corresponds to the serialization point. This point can be determined by inserting a communication statement [ED90] in the global subtransaction, or directly if the operations are submitted separately.

The submission strategy is then:

Order: The global subtransactions are submitted to the LDBMS in an order compatible with the pre-specified order. ❑

Condition2: A global subtransaction will not be submitted to the LDBMS until all previous ones have reached their serialization points. ❑

b)Rigorous schedules

Definition 5.2 [BGMS92]: We say that schedule S is rigorous if, for all pairs of transactions T_i and T_j, if T_i is in indirect conflict with T_j in S and T_j commits in S, then T_j does not execute its conflicting operation before T_i commits. ❑

In this type of schedule, the serialization order corresponds to the validation order. The submission strategy is then:

Order: The global subtransactions are submitted to the LDBMS in an order compatible with the pre-specified order. ❑

Condition3: A global subtransaction will not be submitted to the LDBMS until all previous ones have been committed. ❑

In the three cases, the subtransactions are submitted sequentially when a particular event occurs. A subtransaction waits that this event occurs for all subtransactions that precede it in the pre-specified order. This waiting can be relaxed if we known the set of data items that each subtransaction accesses.

For the second point, two transactions G_i and G_j conflict if there exist two operations $o_i \in G_i$ and $o_j \in G_j$ such that o_i conflicts with o_j, i.e. these two operations access the same data item and one of them is a write operation. The conflict involve a dependency among the two transactions and hence a serialization order. Those two transactions are dependent if they access in conflict the same data. We say that two transactions G_i and G_j access in conflict a common data at a site S_k if:

$O_l(G_i) \cap O_e(G_j) \neq \varnothing$ or

$O_e(G_i) \cap O_e(G_j) \neq \varnothing$ or

$O_e(G_i) \cap O_l(G_j) \neq \varnothing$

Or(T) and Ow(T) are respectively the set of object that are accessed by T in read and write mode respectively. We note the precedent relation:

$D_k(G_i) \cap_c D_k(G_j) \neq \varnothing$ where

$D_k(G_i) = O_l(G_i) \cup O_e(G_i)$ ate site S_k

et $\cap_c$

is the conflict intersection.

The protocol should then enforce a serialization order over global subtransactions that access common data in conflict. It should then guarantee that the dependency among two global subtransactions at a given site do not jeopardize the pre-specified order.

In contrary, if two transactions G_i and G_j do not access in conflict any common data directly (i.e., $D_k(G_i) \cap_c D_k(G_j) = \varnothing$), or indirectly (i.e., there is no global subtransactions $G_1, G_2, \ldots, G_l$ such that $D_k(G_i) \cap_c D_k(G_1) \neq \varnothing$, $D_k(G_1) \cap_c D_k(G_2) \neq \varnothing$, ..., $D_k(G_l) \cap_c D_k(G_j) \neq \varnothing$), then this two transactions can be submitted in an arbitrary order even if the pre-specified order stipulate for example, that G_i must precede G_j. Indeed, there cannot exist any dependency among G_i and G_j (directly or indirectly). The two transactions can be scheduled in an arbitrary manner, any order can exist among them from the point of view of the GTM that ensure the 2LSR. Although the pre-specified order is a total order over global transactions, the order enforced over global subtransactions at each site is partial, it links only global subtransactions that have common data in conflict.

5.2. The Submission Method

We can now state the submission strategy for our protocol.

Order: The global subtransactions are submitted to the LDBMS in an order compatible with the pre-specified order.

Condition: A global subtransaction will not be submitted to the LDBMS until all previous ones that access common data in conflict, have reached their serialization points.

In order to implement the condition, we need a data structure that can interpret two aspects of the submission:

i) the pre-specified order, and

ii) the existence of common data in conflict among global transactions.

The latter aspect can be interpreted by a binary relation that links two transactions with conflict common data. For the former aspect, it suffices to orient this relation. We define then an oriented graph, called **Oriented Conflict Data Access Graph** or **OCDAG**. This graph is defined for each site of multidatabase. $OCDAG_k=(G_k,V_k)$, where G_k is the set of global subtransactions executed at site S_k and V_k is the set of edges such that $G_i \rightarrow G_j$ iff $D_k(G_i) \cap_c D_k(G_j) \neq \varnothing$ et G_i precedes G_j in the pre-specified order. For each node G_i , we associate the following attributes:

• Set of Access Data

• Status: Status of a global subtransaction at site S_k, it has the following values:

Idle: G_j has not yet submitted to the LDBMS.

Before: G_j has been submitted but has not yet:

i) updated the object "SR", or

ii) reached their serialization point, or

iii) been committed.

After: G_j has:

i) updated the object "SR", or

ii) reached their serialization point, or

iii) been committed.

By use of OCDAG, we can reformulate the condition of submission as follows:

Condition: A global subtransaction that arrives at a site S_k or that waits in this site can be submitted if the global subtransactions that precede it in OCDAG have the status equal to After (i.e.: they have all reached their serialization points).

Proof:

We should proof that a global execution E, obtained by our strategy of submission, is 2LSR, i.e.:

i) E_k is serializable for each site S_k, $1 \leq k \leq n$.

ii) The projection of E on G is serializable.

The serializability of local executions is ensured by assumption over the multidatabase. To proof that the projection of E on global transactions is serializable, it suffices to proof that the projections of E_k on G_k (E_k/G_k) are serializable in a compatible order, in our case this is the pre-specified order. We must then proof that the serialization order of all E_k/G_k are compatible with the pre-specified order O. This means that two global subtransactions, G_j and G_i, at the same site S_k, ordered by the pre-specified order O, by example $G_j <_O G_i$, cannot be submitted in a fashion to lead to a dependency, directly or indirectly by solely interposed global subtransactions, that is in contradiction with the pre-specified order O.

Let be G_j and G_i two global subtransactions at site S_k such that:

$G_j <_o G_i$,

G_j arrives first at site S_k, and

G_i arrives after and G_j has not yet submitted.

Three cases are then possible.

First case:

There is a path C_{ji} in $OCDAG_k$ that links G_j to G_i. An inverse path cannot exist by construction since $G_j <_o G_i$. In order to submit G_j it should that at least the global subtransaction that directly precedes G_j in the path C_{ji} has reached its serialization point. Hence, gradually nearer and nearer in the path, G_j can be submitted if G_j has been submitted and has reached its serialization point. The induced dependency on local schedule cannot be incompatible with the pre-specified order, we have always $G_j \rightarrow \ldots \rightarrow G_i$.

Second case:

There isn't a path in $OCDAG_k$ that links G_j to G_i, the two global subtransactions have not common data in conflict. Hence, it cannot exist a dependency among them. In this case, G_j and G_i are not linked, they can be submitted in an arbitrary order. Because the induced dependency, on locale schedule, do not links G_j and G_i, it cannot jeopardise the compatibility of the projection of local schedule on global subtransactions with the pre-specified order.

Third case:

Let G_l be a third global subtransaction that arrives at site S_k after G_i. it follows necessarily G_i and G_j in the pre-specified order, it cannot create a dependencies among them, the only possible dependencies are:

$G_j \rightarrow \ldots \rightarrow G_l$ and/or $G_i \rightarrow \ldots \rightarrow G_l$.

Algorithm:

At each site a server act as an interface between the GTM and the LDBMS. This server must reply to two events:

1) Reception of a global subtransaction from the GTM.

2) Acknowledgement that a certain global subtransaction has reached its serialization point.

The scheduler is then constituted from two procedures:
Procedure **Arrival**(G_i): it is called when a global subtransaction G_i arrives at the site.
Procedure **Waking**(sp_i): it is called when the scheduler know that a global subtransaction G_i has reached its serialization point. This procedure can wake up some other global subtransactions that wait to be submitted by the server.

```
Algorithm Scheduler;
When $G_i$ arrives at site $S_k$
Then
        Arrival ($G_i$)
EndWhen
AND
When receive ack($sp_i$)
Then
        Waking($sp_i$)
EndWhen
End
```

```
Procedure Arrival($G_i$)
Create a node for $G_i$ in $OCDAG_k$;
Status($G_i$)=idle;
For $G_j$ in $OCDAG_k$ such that $D_k(G_i) \cap_c D_k(G_j) \neq \emptyset$
        Do
                Insert ($G_j \rightarrow G_i$) in $OCDAG_k$; /*the
sense of the edge interpret the pre-specified
                                        order*/
        EndDo
/*check to submit $G_i$*/
If ($\forall G_j$ in $OCDAG_k$ such that $G_j \rightarrow G_i \in OCDAG_k$,
Status($G_j$)=After)
Then
        Status($G_i$)=Before:
        Submit($G_i$);
EndIf
 End.
```

```
Procedure Waking($sp_i$);
Status($G_i$)=After;
/*check of waiting global subtransactions */
For $G_j$ in OCDAG such that $G_i \rightarrow G_j \in OCDAG_k$
        Do
        If ($\forall G_l$ in $OCDAG_k$ such that $G_l \rightarrow G_j \in$
$OCDAG_k$, Status($G_l$)=After)
                Then
                        Status($G_j$)=Before:
                        Submit($G_j$);
                EndIf
        EndDo
End.
```

6. Performances

The proposed protocol has many advantages because it adopts the top-down approach for global concurency control. First the flexibility of the scheduler, the global concurrency control is done at local site without global co-ordination by use of the pre-specified order. In addition, it has the following advantages:

No global deadlock.

No inter-sites communication.

No global transactions abortion due to global concurrency control.

In order to point out the interest of the protocol, we compare it with another one described in [MRKS91], we call this latter 2PLG and our protocol TD2L. First, we briefly describe 2PLG.

In the 2PLG protocol, the GTM maintains global locks for global transactions. It then ensures that schedules are 2LSR as follows:

1) global transactions follow the 2PL strategy while obtaining and releasing global locks.

2) A global lock is held by a global transaction at least until the completion of the operation, at the local site, for which the lock was obtained.

We present now an informal comparative study based on various aspects of performances [ED90, VW92]:

Concurrency
Global deadlock
Transaction abortion
Communication
Restrictions

Concurrency

The concurrency of a scheduler S, noted C(S), is defined [ED90] as the set of schedules that can be generated by this scheduler. In a multidatabase, a global schedule is composed of several local schedules. Therefore, the concurrency of the GTM is determined by the concurencies of local schedules. In general, the top-down approach is unable to provide the maximum concurrency. Since the 2LSR order, the pre-specified order, is determined at global level, only those local schedules which are compatible with this order are permitted.

For 2PLG protocol, a global transaction waits the release of global locks on data that it will accede. Since the global locks are managed by a 2PL strategy, a global transaction cannot be submitted until all global transactions that have lock a data that it needs, have reached their global locking points. In TD2L protocol, a global subtransaction waits that all global subtransactions that have common data in conflict with it, have reached their serialization points. The waiting is finer than in 2PLG, since it concerns only the subtransactions not the entire global transaction.

Global deadlock

In the top-down approach a global deadlock cannot occur. The 2LSR order has been determined before the submission of global transactions to local sites, the local schedulers will serialize them in an order compatible with the pre-specified one. A global subtransaction only waits global subtransactions that precede it in the pre-specified order. Since the 2PLG protocol manages the global locks in a 2PL manner, a global deadlock is possible.

Transaction Aborts

Global transactions cannot be aborted, in the TD2L protocol, for the purposes of concurrency control. This is because no global transaction is aborted due to the inconsistency of local schedules. For 2PLG, global transactions may be aborted due to a global deadlock.

Communication

No inter-sites communication is necessary for global concurrency control, except the sending of global subtransactions with the pre-specified order. Again, this is because the 2LSR order is pre-specified at global level. Local schedules can reach an agreement without any communication. The 2PLG protocol needs frequently communication with the GTM and the various sites to manage the global locks.

Restrictions

The 2PLG protocol make no assumptions on local schedules. We need in contrary serialization point based on local schedules. We should review our protocol in general case.

7. Conclusion

The concurrency control in multidatabases is hierarchical due to the local autonomy requirements. Two approaches are then possible to construct a protocol of concurrency control: top-down approach and bottom-up approach. The former have several useful advantages for multidatabases: simplicity of protocol, global deadlock free, ...etc. In contrary, the latter is better in terms of concurrency.

We have then designed a new protocol based on top-down approach that ensure the two level serializability. This protocol should serialize global transactions independently from local ones with respect to a pre-specified order. This is done at each site over global subtransactions by a server process. This process control the submission of global subtransactions by imposing a precedence constraints over only the global subtransactions that can have a dependency relation among them. These are those that access common data in conflict. The submission is then based on serialization points that give the serialization orders of transactions in a local schedule.

The protocol presented in this paper constitutes only one step to design a complete multidatabase transaction manager, several other aspects can be shown as perspectives. In our study, we have omitted to mention the moment when a node of OCDAG is removed. In fact, this will be done by the validation-recovery module. This module should ensure atomicity and durability of global transactions in presence of failures. Another perspective is to extend the protocol to other type of schedules that are not based on serialization points. Nevertheless, the assumption over local schedules is not restrictive since most commercial DBMS (e.g.: 2PL) produce serialization point based schedules. It is interesting to note also that a top-down approach permits to use the appropriate process server at each site , since the co-ordination among the various servers is done by the pre-specified order.

References

[AGMS87] R.Alonso, H.Garcia-Molina, and K.Salem.
Concurrency control and recovery for global procedures in federated database systems.
In IEEE Data Eng. Bulletin, pp. 5-11, September 1987.

[BGH87] P.Bernstein, V.Hadzilacos, and N.Goodman.
Concurrency Control and Recovery in Databases Systems.
Addison-Wesley Publishing Co., 1987.

[BGMS92] Y.Breitbart, H.Garcia-Molina and A.Silberschatz.
Overview of Multidatabase Transaction Management.
Dept. of Computer Science, Stanfors University, Report N°. STAN-CS-92-1432, May 1992.

[BS88] Y.Breitbart and A.Silberschatz.
Multidatabase update issues.
In Proc. of ACM SIGMOD Intl. conf. on Management of Data, June 1988.

[DE89] W.Du and A.K.Elmagarmid.
Quasi Serialisability: a Correctness Criterion for Global Concurrency control in Interbase.
In Proc. of the 15th Intl. conf. on Very Large DataBases, pp.347-355, Amesterdam 1989.

[DEK91] W.Du, A.K.Elmagarmid and W.Kim.
Maintaining Quasi Serializability: in Multidatabase System.
In Proc. of the 7th Intl. conf. on Data Eng. (IEEE), pp.360-367, Japan 1991.

[ED90] A.K.Elmagarmid and W.Du.
A Paradigm for Concurrency Control in Heterogeneous Distributed Database Systems.
In Proc of the Sixth Intl. Conf. on Data Eng, pp.37-46, 1990.

[GM91] H.Garcia-Molina.
Global Consistency Considered Harmful for Heterogeneous Database Systems.
In Proc. of 1st Intl. Workshop on Interoperability in Multidatabase Systems (IMS91), Kyoto, Avril 91.

[GRS91] D.Geogakoupoulos, M.Rusinkiewicz, and A.Sheth.
On Serializability of Multidatabase Transactions through Forced Local Conflicts.
In Proc. of the 7th Intl. conf. on Data Eng. (IEEE), pp.360-367, Japan 1991.

[MRBKS92] S.Mehrotra, R.Rastogi, Y.Breitbart, H.F.Korth and A.Silberschatz.
The Concurrency Control Problem in Multidatabases: Characteristics and Solutions.
In Proc. of the 1992 ACM SIGMOD Intl. Conf. on Management of Data, pp. 288-297, San Diego, CA, 1992.

[MRKS91] S.Mehrotra, R.Rastogi, H.F.Korth and A.Silberschatz.
Non-Serializable Executions in Heterogeneous Distributed Database Systems.
In Proc. of the 1st Intl. Conf. on Parallel and Distributed Information Systems, Miami-Beach, FL, December 1991.

[Pu88] C.Pu.
Superdatabases for composition of heterogeneous databases.
In Proc. of the Intl Conf on Data Eng, pp. 548-555. February 1988.

[SL90] A.Sheth and J.Larson.
Federated Database Systems for Managing Distributed. Heterogeneous, and Autonomous Databases.
ACM Computing Surveys, Special Issue on Heterogeneous Databases, Vol. 22, No. 3, pp. 183-236, Sept. 1990.

[Sug87] K.Sugihara.
Concurrency control based on cycle detection.
In Proc. of the Intl. Conf. on Data Eng., pp. 267-274, February 1987.

[VW92] J.Veijalainen and A.Wolski.
Prepare and Commit Certification for Decentralized Transaction Management in Rigorous Heterogeneous Multidatabases.
In Proc. of the 8th Intl. Conf. on Data. Eng. Phoenix, pp. 470-479, AZ 1992.