An Approach for Ensuring Concurrency Control of Global Transactions in Heterogeneous Distributed Database Systems

Arun Kumar Yadav[1], Dr. Ajay Agarwal[2], Kamlesh Chopra[3]

[1]Associate Professor, Venkteshwar Institute of Technology, Indore (M.P.), India
[2] Professor & Head, Krishna Institute of Engg. & Technology,Ghaziabad (U.P.), India
[3]Lecturer, Venkteshwar Institute of Technology, Indore (M.P.), India

Abstract
In this paper, we have proposed the strategies for concurrency control of global transactions in heterogeneous distributed database systems. We have focused on the issues of consistency, local autonomy and performance. A technique to prevent and resolve inconsistency, classified into one of the two basic approaches: optimistic or pessimistic. The previous research intends to provide a high degree of concurrency among global transactions, while the later is concerned with aborts of global transactions. The strengths and weaknesses of the two approaches are discussed.

Keywords:
Heterogeneous Distributed Database System (HDDBS), Local Database System (LDBSs), Local Concurrency Controller (LCC), Global Concurrency Controller (GCC),

1 Introduction

A heterogeneous distributed database system (HDDBS) is a group of pre-existing database systems (called Local Database Systems or LDBSs). An HDDBS is the natural result of shifting priorities and needs of an organization as it acquires new database systems that are designed independently. For many applications, an HDDBS is an attractive alternative to a single, integrated database system. An HDDBS is different from a set of database systems in that it supports global applications accessing mu1tiple systems simultaneously. It is also different from traditional homogeneous distributed database systems in that it interconnects LDBSs in a bottom up fashion, thereby allowing existing applications developed on each of the LDBSs continue to be executing without any modification.

One of the important feature of HDDBSs is autonomy of LDBSs. Local autonomy defines the right of each LDBS to control access to its data by other LDBSs and the right to access and administer its own data independently of other LDBSs. As a result, LDBSs may use different data models, different concurrency control strategies and they can schedule to accesses its data independently. Local autonomy is required and necessary in HDDBSs to guarantee that old applications are executable after interconnection, to facilitate flexible interconnection of various kinds of LDBSs, and to ensure the consistency and the security of LDBSs.

Concurrency control in HDDBSs is different from homogeneous distributed database systems, due to existence of local concurrency controllers (LCCs). An LCC reside at each LDBS and maintains its consistency. LCCs are not capable of maintaining the consistency of the global database, because global transactions may be scheduled inconsistently at different sites. In order to prevent this kind of inconsistency, a global concurrency controller (GCC) is needed. The GCC is built on top of LCCs coordinating local executions at different sites.

Controlling the concurrency in HDDBSs is more difficult than that in homogeneous distributed database systems, due to the autonomy of LCCs [1]. The LCCs are independently designed and cannot be modified because of the autonomy restrictions placed by the LDBSs. In addition, the LCCs have the right to schedule local and global transactions independently, based on their own considerations. The GCC has only control over local executions in submissions of global transactions. However, it is possible that a global transaction effectively precedes another even if it is executed entirely at all local sites [2].

The necessity and difficulties of the global concurrency control in HDDBSs were accepted in [3]. Several conditions for global concurrency control were identified in [4]. Since then, a large amount of work has been done in developing algorithms for controlling global concurrency.

In this paper, we are presenting algorithms to preserve local autonomy, maintaining the global consistency and will discuss their strengths, weaknesses, and performance. The rest of the sections are organized as follows. We first discuss, in Section 2, three basic issues of global concurrency control in HDDBSs, namely, autonomy, consistency and performance. In Sections 3 and 4, we discuss the solutions for global concurrency control. Section 5 concludes the paper with summary of algorithms, results and suggestions for future work.

Manuscript received March 5, 2010
Manuscript revised March 20, 2010

2. Local Autonomy, Consistency and Performance

The concurrency control is needed to maintain database consistency and provide high performance by allowing concurrent execution of transactions. In HDDBSs, the global concurrency control mechanisms should also preserve the autonomy of LDBSs. In this section, we are discussing the three issues (i.e., local autonomy, consistency and performance) in the context of HDDBS. Firstly we are presenting an HDDBS model. A classification of global concurrency control strategies is also presented in this section.

2.1 HDDBS Model

An HDDBS is a distributed database system consisting of LDBSs. Each local database is a set of data items. There are two kinds of transactions in an HDDBS. A local transaction accesses data items of one local database, while a global transaction accesses data items of more than one local database. A global transaction (consists of a set of global sub-transactions) accesses a single local database and is executed at the sites along with local transactions. When local and global transactions are executed, they transform the database from one consistent state to another consistent state.

More specifically, a global transaction transforms the global database from a consistent state to another consistent state, while a local transaction and a global sub-transaction transforms a local database from a consistent state to another consistent state.

The transaction processing model for HDDBSs is shown in Figure 1. It consists of a Global Data manager (GDM), a Global transaction Manager (GTM), a collection of LDBSs, a set of global and local transactions and a set of server processes. A server process runs at each site and represents the interface between the GTM and the LDBSs. Global Data manager (GDM) decomposes the global transaction into a set of sub-transactions and a Global transaction Manager (GTM) submits the sub-transactions to the corresponding LDBSs for the execution.

When a global transaction is submitted to the HDDBS, it first decomposes and translated by the GDM into a set of sub-transactions that run at the local sites where the required data reside. It is assumed that, for every global transaction, there is at most one sub-transaction per site [3]. The GTM submits the sub-transactions to the corresponding LDBSs and coordinates their executions at local sites so that the global database consistency is maintained. The GCC is a module of the GTM which is responsible for the g1obal concurrency control.

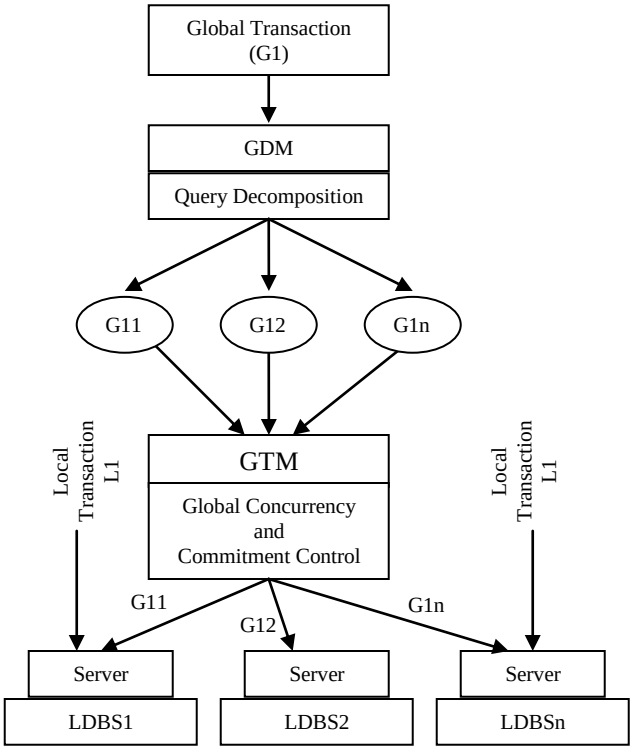


Figure 1: A Transaction Processing Model

These sub-transactions are then directed to the GTM.

2.2 Local Autonomy

Local autonomy is an important feature of HDDBSs. It defines the ability of each LDBS to perform various kinds of operations and various kinds of control over its own database. For example, an LDBS should be able to implement its own data model, catalog management strategy, and its own naming convention. It should also be able to control over local transactions and global sub-transactions (e.g., to delay or abort a transaction) to maintain the consistency of the local database. Local autonomy also defines the right of each LDBS to make decisions regarding the service it provides to other LDBSs. Local autonomy is required to guarantee that local users can continue to run local applications on their LDBS regardless of interconnection and to ensure that the basic consistency, security and performance requirements of an LDBS are met while allowing other LDBSs to access its data.

It is difficult to compute local autonomy. To discuss how well a global concurrency control strategy preserves local autonomy, we distinguish among different aspects of local autonomy. The ability of global concurrency control strategies to maintain these aspects of autonomy is discussed in the next two sections.

2.3 Consistency

As, in homogeneous distributed database systems, the goal of global concurrency control in HDDBSs is to maintain the HDDBS consistency. Due to the hierarchical structure of HDDBSs, two kinds of consistency exist: one that previously existed in each of the LDBSs (called local consistency) and the other that results from the interconnection process (called global consistency). Local consistency defines constraints on relationships of data in a local database and on interactions among local transactions and global sub-transactions executed at the site, while global consistency defines constraints on relationship of data at different local database and on interactions among global transactions, as well as interactions among local transactions executed at different sites. We assume that the LCC at each site maintains the local consistency of the LDBS. More specially, it guarantees the serializability of the local execution. It is the GCC's responsibility to coordinate local execution to ensure global consistency.

The conventional way to maintain the global consistency is to execute global and local transactions in a serializable fashion. An execution of a set of transactions is serializable if it is equivalent to a serial execution of the transactions. Serializability in HDDBS represents the strongest type of consistency in that it treats an HDDBS as a strongly coupled homogeneous distributed database system because GCC ensures that the concurrent execution of global sub-transactions should be equivalent to the serial execution. There is no difference between local and global transactions and between local and global consistencies. The problem with serializability in HDDBS is that it is difficult to maintain, due to local autonomy of LDBs ([5], [1]). Since LCCs have the right to schedule local transaction and global sub-transactions independently, the only control the GCC has over local executions is the submission of global transactions. As mentioned before, it is generally impossible to maintain the global serializability by just controlling the submission of global transactions.

2.4 Performance

Degree of concurrency is the main measure of performance for concurrency control in database systems. The degree of concurrency refers to the number of transactions executing concurrently and provided by a concurrency control protocol and is measured by the possible interleaving of transactions it allows. In HDDBSs, we consider two kind of interleaving: that among global transactions and that among local transactions and global sub-transactions at a site. The latter is determined by LCCs. The GCC is responsible for providing a high degree of concurrency of global transactions. One way of achieving the goal is to impose few restrictions as possible on submission and execution of global transactions.

Another important issue affecting performance is abort ratio (i.e. the number of transactions selected as victim to terminate from the system) of global transactions. In HDDBSs, global transactions are aborted not only because of the conflicts in a local execution, but also because of the inconsistency of serialization orders between local executions. Consider two global transactions G_1 and G_2 which access two local databases L_1 and L_2 respectively. Suppose that they are submitted at about the same time t_1. Due to the different scheduling policies of two LDBSs use, it is possible that the two global transactions are scheduled in different orders at two sites L_1 and L_2. Generally, the more common sites two global transactions access, the more likely that they will be scheduled inconsistently. Due to the same submission time of global transactions G_1 and G_2, one of the global transactions may be abort to ensure the consistency of the database.

Another reason that abort ratio of global transactions is important in HDDBSs is that the execution of a global transaction is usually expensive. The abort of a global transaction implies that all its sub-transactions must be aborted. This is very difficult at some sites due to local autonomy.

Therefore, it is very important for a concurrency control strategy to abort as few global transactions as possible. This can be done in two ways. First, it should try not to abort global transactions because of local conflicts. In other words, the abort of a sub-transaction should not result in the abort of the global transaction. Second, it should reduce the possibility of inconsistency between local executions, and in case an inconsistency occurs, aborts no more global transactions than necessary.

Providing the high degrees of concurrency and minimizing the number of global transactions aborted is an important decision in designing a global concurrency control algorithm, and results in two different approaches of global concurrency control.

2.5 Classification of Strategies

A global concurrency control strategy in HDDBSs can be classified as pessimistic and optimistic. The optimistic approaches try to provide a high degree of concurrency, while the pessimistic approaches are concerned with the abort ratio of global transactions.

Generally, pessimistic approaches attempt to prevent inconsistency by imposing restrictions on the submission of global transactions. The GCC assumes that there are conflicts among global transactions whenever such conflicts possibly exist. The strategies in this group differ in the restrictions imposed on the submission of global transactions to guarantee a specific (serialization or quasi

serialization) order. GCCs based on this approach do not abort global transactions due to inconsistency between executions. On the other hand, they allow low degrees of concurrency because before submitting the global transactions it ensures the serialization or quasi serialization. In addition, they are usually unable to make use of dynamic information about local executions because it is impossible to predict in advance whether the information is available.

In contrast to pessimistic approaches, optimistic approaches do not control the submission of global transactions. Instead, they detect and resolve inconsistency after the execution of global transactions. The approaches are based on the assumptions that not every pair of global operations could conflict arbitrarily. The strategies in this group differ in how they detect and resolve inconsistency.

3. Pessimistic Approaches

In this section, we discuss four pessimistic approaches of global concurrency: altruistic locking, top-down, site graph and access graph. One property they all share is that no global transaction is aborted due to inconsistency among local executions. Therefore, the main measure of performance is the degree of concurrency of global transactions.

3.1 Altruistic Locking

The altruistic locking strategy was initially proposed for long-lived transactions and is also used as a concurrency control strategy in HDDBSs [6]. The basic idea of the strategy is to prevent inconsistency by not allowing two global transactions to access a local database concurrently. A global transaction must lock a site before it can access the site. Once its all requests have been processed, it can release the lock on the site. Releasing a lock is a conditional unlocking operation. Other global transactions waiting to lock the released site may be able to do so if they are willing to abide by the following rules. (1) No two global transactions hold locks on the same site simultaneously unless one of the transactions locked and released the site before the other locked it. In other words, the later lock-holder is in the wake of the releasing transaction. (2) If a transaction is in the wake of another transaction, it must be completely in the wake of the transaction. In other words, if G1 locks site A which has been released by G2, then anything currently locked by G1 must have been released by G2 before it was locked by G1. (This requirement is relaxed once G2 finishes.) Note that the releasing operation is not two-phase, i.e. the global transaction is free to continue to lock new sites after it has released locks; while the unlocking operations is two-phase and usually is done after global transactions complete.

Example 1. Consider an HDDBS consisting of five LDBSs: A, B, C, D, and E. Let G1 be a global transaction which accesses sites A, C, D, and E in that order, G2 accesses sites A and C, and G3 accesses sites A and B. Suppose that at current state, G1 hold the locks on sites A, C and D. The sites A, C and D constitute the current wake of G1. After G1 released the locks, G2 is allowed to access sites A and C, since they are in the wake of G1.G3, however, can only access site A, and has to wait until G2 completes and unlocks all its locks before it can access site B, which is outside the wake of G1.

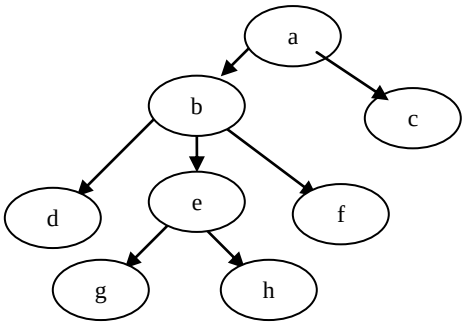


Figure 2: A Rooted tree interconnected HDDBS

The execution order of global transactions is decided in the following way. If a global transaction is in the wake of other global transactions, then its order is after those global transactions. For global transactions which are not in the wake of each other, their order is decided by the order they lock the conflict sites. It is shown that the altruistic locking maintains the same execution order among global transactions at all sites.

The main advantage of this strategy is that it exercises no control over and requires no information about local executions. In other words, it preserves local autonomy of LDBSs. However, the strategy provides no concurrency among global transactions because they are executed in serial manner at each site. The benefit of doing this is that the execution orders of global transactions at local sites are always consistent. Another problem is that it maintains global serializability only if all LCCs maintain strict serializability of local executions.

In other words, serialization order of global sub-transactions is compatible with their execution order. This is however not true for concurrency control protocols (e.g., serialization graph testing). On the other hand, it has been shown in [7] that it maintains quasi serializability regardless of concurrency control protocols LCCs use.

3.2 Top-down

As discussed in [4], one way of performing global concurrency control in HDDBSs is the top-down approach. In top-down approach, the GCC determines the execution order of global transactions before their submission to

local sites which is then enforced at each site. There are two basic steps in a top-down approach of concurrency control: (1) determining an order at global level, and (2) enforcing the order at local level.

The non-two-phase locking protocol proposed in [8] gives a solution in the first step. It provides a way of dynamically determining the order as global transactions are executed. This approach is adapted from the well known non-two-phase locking protocol for the traditional concurrency control problem [9].

The protocol applies to those HDDBSs that are formed as a rooted tree. The sub-transaction of a global transaction is treated as an atomic transaction step, and each LDBS is treated as a distinct data item. The GCC issues the sub-transactions of a global transaction to the individual LDBSs according to the tree protocol. The following is a simple example illustrating the idea.

Example 2. Consider an HDDBS which is interconnected as a rooted tree in Figure 2. Let G be a. global transaction and G_x, be its sub transaction executed at site X. Suppose that sub-transactions of G are to be executed at sites B, E, F and H. It first "locks" B and send sub- transaction G_b to site B. Then it locks E and F. After that, and after the execution of G_b completes, it may release the lock on B. Likewise, on completion of the execution of G_e, and after locking H, it can release the lock on E. While the locks on sites are acquired in the tree order, it can be released in any order.

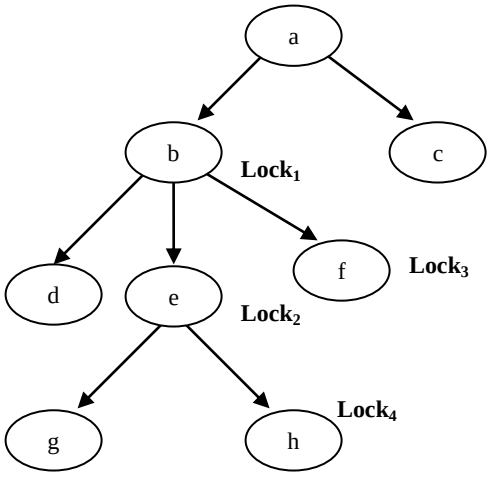


Figure 3: Top-Down locking in rooted tree interconnected HDDBS

In [10], various techniques of enforcing the global execution order at local sites are discussed. One way of doing this is to control the submission order of global sub transactions. The task is performed by server processes at local sites (called stub processes in [10] and site queue in [11]). The server process guarantees that the LCC schedules global sub-transactions in the order they are submitted. More specifically, the server process submits a global transaction to the LCC only if the serialization orders of all previously submitted global transactions have been determined by the LCC. The technique works for those LCCs that determine the serialization order of a transaction according to an event which can be identified before the termination of the transaction. An LCC with this property is said to be static [11].Both two phase locking and time-stamp ordering protocols are static. For example, in two-phase locking protocols, the serialization order of a transaction is determined once it reaches its lock point.

Unfortunately, not all concurrency control protocols are static, e.g., serialization graph testing. It is generally impossible to guarantee a specific serialization order using the above technique. On the other hand, the technique can always be used to impose a quasi serialization order. The degree of concurrency provided by the algorithm depends on local concurrency control strategies. For example, if two-phase locking protocol is used, global sub-transactions at the site are executed almost sequentially. A higher degree of concurrency is possible if timestamp ordering protocol is used. The price is, however, possible conflicts between global sub-transactions, and therefore aborts of some sub-transactions.

3.3 Site Graph

Site graph algorithm proposed in [12] is an attempt to maintain global serializability without imposing any restriction on LCCs. In this algorithm, the GCC maintains an undirected graph, called site graph, in which the nodes are sites and the edges are global transactions. When a global transaction T is received, the GCC first determines the sites that contain copies of data accessed by the global transaction and connects them to form a linear link in the graph. The following example illustrates the notions of site graphs:

Example 3. Consider an HDDBS that contains data item x at sites 1 and 2, y at sites 1 and 3, and z at sites 2 and 3. Let G_1 and G_2 be two global transactions:

G_1:$r_1(x)$ $w_1(y)$ G_2:$r_2(y)$ w_2 (z)

Since the data are replicated, the GCC may generate one of the following sequences of local operations for each global transaction:

G_1:$r_1(x_1)$ w_1 (y_1) w_1 (y_3) G_2:r_2 (y_3) w_2 (z_2) w_2 (z_3) or

G_1:$r_1(x_1)$ w_1 (y_1) w_1 (y_3) G_2:r_2 (y_1) w_2 (z_2) w_2 (z_3)

The site graphs for the sequences are shown in Figure 4. (a) and 4. (b).

In Figure 4, site graph 4. (a) is acyclic, while site graph 4.(b) contains a cycle. That a site graph is cyclic means that the execution of the global and local transactions may be non-serializable. For instance, the following non-serializable execution may occur if the second sequence is submitted for execution.

Site 1: $r_1(x_1)$ w_1 (y_1) r_2 (y_1)

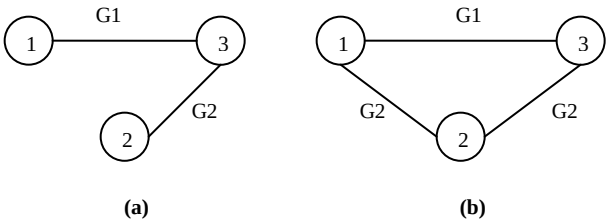


Figure 4: The Site Graph of G1 and G2

Site 2: w_2 (z_2)
Site 3: r_1 (y_3) w_1 (y_3) w_2 (z_3) w1 (z_3)
In this execution, local transaction L introduces an indirect order between G1 and G2 at site 3, which is different from their order of conflicting operations at site 1. While for the first sequence, the local transaction can not introduce indirect orders and therefore is guaranteed to result in a serializable execution.

The algorithm is the only pessimistic strategy that both maintains global serializability and does not violate local autonomy. However, it allows a lower degree of concurrency among global transactions. The reason is that the edges in a site graph cannot be purged even after the completion of the corresponding transaction, as illustrated by the fo1lowing execution.
E_l:
$r_1(a)w_{g1}(a)w_{g2}(b)w_{11}(b)r_{12}(c)w_{g2}(c)w_{g3}(d)w_{12}(d)….r_{1n}(y)w_{gn}(y)w_{gn+1}(z)w_{1n}(z)$

In this execution, G1 executes entirely before G2, which executes entirely before G3 and so on. The serialization order, however, is the reverse: $G_{n+1}\rightarrow G_n\rightarrow…..G_2\rightarrow G_1$.In other words, a global transaction (e.g., G_{n+l}) may effectively precedes another global transaction (e.g., G_1) which executed arbitrarily long before.

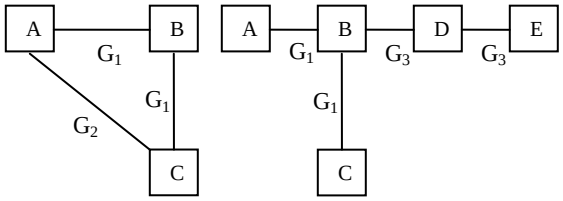
3.4 Access Graph

The basic idea of the access graph algorithm proposed in [13] is similar to the site graph algorithm. The algorithm provides a higher degree of concurrency than the site graph algorithm. More specifically, it only guarantees quasi serializability in general.

The algorithm is based on the following property of quasi serializable executions. The quasi serialization order of two global transactions is compatible with their execution order if they do not interleave with each other. We say that two transactions interleave with each other if either they are executed concurrently (direct interleaving) or a third transaction directly interleave with both of them (indirect interleaving). The algorithm ensures the quasi serializability of a global execution by maintaining the acyclicity of the access graph of the execution. An access graph characterizes the current execution environment from a single global transaction's point of view. More specifically, the access graph of an execution with respect to a global transaction is a sub-graph of its site graph. The sub-graph consists of only those edges that are introduced by global transactions who interleave (either directly or indirectly) with the global transaction. It has been shown that the quasi serializability of an execution is assured if its access graphs are acyclic with respect to all global transactions.

The algorithm works by grouping global transactions. The transactions in each group form an acyclic access graph and therefore can be submitted to local sites without any control. Transactions at different groups, however, must be executed separately.

Example 4. Consider an HDDBS consisting of five LDBSs: A, B, C,D, and E. Let G1, G2 and G3 be three global transactions where G1 accesses A, B and C, G2 accesses A and C, and G3 accesses B, D and E. Suppose that they are submitted to the GCC in the order: GI,G2and G3. The GCC directs G1 to the local sites immediately. G2, however, cannot be submitted until G1 finished because they form a cyclic access graph. Since G1 and G3 do not form a cyclic access graph, G3 is submitted to local sites and executed concurrently with G1 at site B. After both G1 and G3 have finished, G2 will be submitted.



5. a Cyclic Access Graph **5. b Acyclic Access Graph**

Figure 5 Access Graph

The main advantage of using quasi serializability as the correctness condition is that the GCC only needs to consider those global transactions that are in the same group. Therefore, edges in an access graph are purged after all global transactions in the corresponding group have completed. As a result, it provides a higher degree of concurrency than the other three algorithms. In addition, it does not violate local autonomy. On the other hand, it is only suited to those HDDDS applications where quasi serializability is appropriate as a correctness criterion.

3.5 Discussion

The basic idea of pessimistic approaches is to prevent inconsistency between local executions from occurring. There are two ways of doing this. The first is to control the order in which global sub-transactions are submitted, and the second is to control possible interactions among global

transactions. The altruistic locking and top-down algorithms follow the first approach, while the site graph algorithm follows the second. The access graph algorithm combines the two approaches and it controls interactions among global transactions in a group and controls submission order of global transactions in different groups.

The main advantage of the first approach is the relatively high degree of concurrency (comparing to the second one), due to making use of static information of LCCs (i.e., when and how an LCC determines the serialization order of a transaction). For example, the altruistic locking algorithm allows a global transaction to access a site which is still locked by other transactions as long as they request no more work at the site. Similarly, the top-down algorithm allows a global transaction to access a site after the previous transactions serialization orders have been determined. So a basic assumption behind the approach is that LCCs serialize global sub-transactions in an order compatible with their submission order. This, generally, implies violation of static autonomy.

The second approach (controlling possible interactions) makes no assumption on LCCs in maintaining global serializability. Therefore, it can be applied to all HDDBS applications. As mentioned before, the approach suffers from low degree of concurrency.

The problems with both approaches may be solved if global serializability is not required. For example, if quasi serializability is used as the correctness criterion, the assumption of compatibility of submission order and serialization order is not necessary in the first approach, and the degree of concurrency in the second approach can also be greatly improved.

4. Optimistic Approaches

The three algorithms discussed in this section share a common property of imposing no restriction on submission of global transactions. Theoretically, they can provide a very high degree of concurrency among global transactions. In this approach the aborts of global transactions is possible. Due to the difficulties of detecting inconsistency, they may abort more global transactions than necessary. All three approaches use serializability as the standard.

4.1 Decentralized Global Concurrency Control

A decentralized global concurrency control algorithm for HDDBSs is proposed in [14] where each LDBS uses two-phase locking strategy. The basic idea of the algorithm is to maintain global serializability by synchronizing release of locks held by sub-transactions of a global transaction. A global sub-transaction will not release locks until it receives an "end-of-transaction" message issued by the GCC. The message is issued only if the global transaction finishes its execution at all local sites. Semantically, the "end-of-transaction" is equivalent to the "commit" message in two-phase commit protocol. It has been shown that global serializability is assured if all global transactions follow this protocol.

On the other hand, global deadlocks may occur, due to inconsistency among local executions. Consider two conflicting global transactions G1 and G2 that both access sites 1and 2. Suppose that G1 is scheduled before G2 at Site 1, but after G2 at Site 2. Then G2 waits for locks held by G1 at Site 1, which in turn waits for locks held by G2 at Site 2. The algorithm is optimistic in the sense that it detects inconsistencies (global deadlocks in this case) after the execution. Two algorithms have been proposed for deadlock detection. The first is a time-out based mechanism which checks the acyclicity of the potential conflict graph (PCG) of an execution. A PCG is a directed graph G = (V, E), where V consists of a set of global transactions and E consists of edges $G_i \rightarrow G_j$ such that G_i is in the waiting state and G_j in the active state at a local site. The acyclicity of PCGs guarantees there is no global deadlock. The cyclicity of PCGs, on the other hand, there may or may not be deadlocks. The second algorithm is based on the concept of value date [15]. The GCC associates a value date (timestamp) with each global transaction which denotes when the transaction is supposed to finish. Global transactions that pass their value date are aborted by the GCC, regardless of whether they are still running. Global deadlocks, therefore, are always broken after some of global transactions reach their value dates.

The algorithm makes a practical assumption in that it requires all LDBSs to use two-phase locking. This, from a strict sense, violates design autonomy of LDBSs. No LDBS is allowed to join the HDDBS unless it uses two-phase locking as the concurrency control strategy.

In addition, both deadlock detection algorithms may detect global deadlocks that do not exist, resulting aborts of more global transactions than required. On the other hand, it maintains global serializability and provides a high degree of concurrency.

4.2 Super Database

An optimistic approach has been proposed in [16] called the Super Databases. The basic idea behind this approach is as follows. Each LCC executes the sub-transactions as an ordinary local transaction. When a sub-transaction is completed, the corresponding LCC reports to the GCC the serialization order called 0-element. The GCC then constructs for each global transaction an order-vector (0-vector) as the concatenation of all 0-elements of the sub-transactions of the global transaction. The order on the 0-vectors is defined in a strict sense. 0-vector (T_1) →0-vector (T2) if and only if for all LDBSj, 0-element (T_1, j)

→0-element (T_2, j). If a global transaction does not have a sub-transaction at a particular site, a wild-card 0-element, denoted by * (star) is used for the corresponding component of the 0-vector. The order of this component does not matter and, by definition, 0-element (any) →*, and, * →0-element (any). A certification is done for a global transaction when it tries to commit. It is done by comparing the 0-vector of the committing global transaction against the 0-vectors of the recently committed global transactions. If the new 0-vector can find a place in the total order of the recently committed global transactions 0-vectors, it is committed; otherwise, it is aborted.

It is not hard to see that the algorithm maintains global serializability. It also provides a high degree of concurrency because it aborts only those global transactions that introduce inconsistency. In addition, it preserves static autonomy in the sense that it imposes no restriction on local concurrency control strategies that LDBSs use.

However, two minor issues must be addressed before this algorithm can be widely used. The first problem is that the LCC may not be able to determine the serialization order of a transaction even after the completion of the transaction. Generally, the serialization order of a transaction depends not only on the execution of previously executed transactions but also on the execution of those executed later. The second problem is that it is not clear how the GCC can get 0-elements of global sub-transactions from LDBSs.

4.3 Optimistic Timestamp

The optimistic Timestamp approach proposed in [17] is motivated by the observation that it is generally easier to predict serialization order between global transactions that directly conflicts with each other. The approach introduces a synchronization object called timestamp in each site, and requires each sub-transaction to access the timestamp on its site. Each access to the timestamp consists of the following two operations: reading the current value and incrementing it by one. These two operations must be included in the code of each sub-transaction and are synchronized along with other database read/write operations (so that access to these two operations is atomic). It is assumed that each LCC maintains the serializability of its local execution. According to the serializability theory, two conflict transactions are ordered in accordance with the order of their conflicting operations. Since the timestamp accessing operations are conflicting operations, the serialization order of the sub-transactions is the same as the order that their timestamp accessing operations are performed. Furthermore, the timestamp are monotonously increased when it is accessed. Therefore, the timestamp value accessed reflects the order that the timestamp accessing operation is executed, and therefore reflects the serialization order of the sub-transaction. The serialization order, after being obtained, can be used by the GCC to validate the execution of the global transactions. The timestamp accessing operations are embedded in the code of the sub-transactions. No explicit serialization order is needed from the LCC; therefore, no modification of the LCC is required.

The algorithm is interesting and unique in sense it maintains the global serializability, provides a high degree of concurrency and does not violate local autonomy. The algorithm addresses the problem of how to effectively obtain the serializations orders of global transactions. A minor problem with this algorithm is that it may abort more global transactions than required, due to conflicts introduced by timestamp. Since every two global transactions directly conflict with each other at each site they access, it is very likely that there is a cycle in the global serialization graph, unless some restrictions are imposed on their submission.

5. Conclusion

5.1 Summary and Results

A brief summary of seven algorithms discussed in the paper is given in the following table.

		Consistency	Autonomy	Concurrency	Exact Abort
Pessimistic	Altruistic locking	SR/QSR	Design/√	6	1
	Top-Down	SR/QSR	Design/√	5	1
	Site Graph	SR	√	7	1
	Access Graph	SR/QSR	Design/√	4	1
Optimistic	Decentralized GCC	SR	Static	1	6
	Super Database	SR	Dynamic	1	5
	Optimistic Ticket	SR	√	1	7

Table 5.1

In the Table 5.1**,** an SR entry in the third column indicates that the algorithm maintains global serializability, while an SR/QSR entry indicates that the algorithm maintains global serializability in some environments, and maintains quasi serializability in general. A check mark √ in the fourth column means that the algorithm preserves local autonomy, while Static (or Dynamic) means that the algorithm may violate static (or dynamic) autonomy. A Design / √ entry means that the algorithm may violate design autonomy if serializability is used as the

correctness criterion, but preserves local autonomy if quasi serializability is used as the criterion. The concurrency degree and transaction abort ratio are numbered in order: 1for the best and 7 for the worst.

Generally, optimistic approaches are superior to pessimistic approaches in concurrency degree, but inferior to in transaction abort ratio. In most HDDBSs applications, the number of global transactions is small in comparison to that of local transactions. Therefore, a low degree of concurrency among global transactions does not affect the overall concurrency very much. On the other hand, a high global transaction abort ratio is usually unacceptable in most HDDBS applications.

One problem with most pessimistic approaches is that they maintain global serializability only if the LDBSs use specific concurrency control strategies. The problem, however, is not that bad in practice. The reason is that the algorithms work for most practically used strategies (e.g., two-phase locking and timestamp ordering protocols).

The key point in success of optimistic approaches is to detect inconsistency between local executions precisely. Although it is very likely that global transactions are scheduled inconsistently at different sites, they usually do not conflict with each other. If the real conflicts can be detected precisely, the approaches will provide good performance. Unfortunately, this is difficult, due to local autonomy.

5.2 Future Work

Global concurrency control is still an active research area. The following are some research directions that we think for future.

Pessimistic and optimistic approaches each have advantages over the other in some aspects. An interesting problem is therefore how to combine them together to develop an algorithm that not only provides a high degree of concurrency but also aborts a small number of global transactions.

As in many other research areas, one deficiency in our discussion of global concurrency control is the lack of performance data. Little work has been done in this important area.

Another problem with the existing concurrency control strategies is that most of them are designed independently of other transaction management issues, e.g., commitment and recovery. Clearly, more attention should be devoted to these issues, as well as the integration with concurrency control strategies.

References

[1] W. Du, A. Elmagarmid, and W. Kim. Effects of local autonomy on heterogeneous distributed database systems. Technical Report ACT-00DS-EI-050-90, MCC, February 2000.

[2] P. Bcrnstein, V. Hadzilacos, and N. Goodman. Concurrency Control and Recovery in Databases Systems. Addison- Wesley Publishing Co., 2002.

[3] V. Gligor and G. Luckcenbaugh. Interconnecting heterogeneous data base management systems. IEEE Computer, 17(1):33-43, January 2000.

[4] V. Gligor and R. Popescu-Zeletin. Transaction management in distributed heterogeneous Database management systems. Information Systems, 11(4):287-297, 2004.

[5] W. Du, A. Elmagarmid, Y. Leu, and S. Ostermann. Effects of autonomy on global concurrency control in heterogeneous distributed database systems. In Proceedings of the Second International Conference on data and Knowledge Systems for Manufacturing and Engineering, pages 113-120, Gaithersburg, Maryland, October 2004.

[6] R, Alonso, H. Garcia-Molina, and K. Salem. Concurrency control and recovery for global procedures in federated database systems. In IEEE Data Engineering Bulletin, pages 5-11, September 2006.

[7] W. Du and A. Elmagarmid. Quasi serializability: a correctness criterion for global concurrency control in InterBase. In Proceedings of the International Conference on Very Large Data Bases, pages 347-355, Amsterdam, The Netherlands, August 2000.

[8] K. Vidyasankar. Non-two phase locking protocols for global concurrency control in distributed heterogeneous database systems. In CIPS, 2000.

[9] A. Silberschatz and Z. Kedem. Consistency in hierarchical database systems. Journal of the ACM, 27(1):72-80, 2002.

[10] A. Elmagarmid and W. Du. A paradigm for concurrency control in heterogeneous distributed database systems. In Proceedings of the Sixth International Conference on Data Engineering, Los Angeles, California, February 2004.

[11] Y. Leu and A. Elmagarmid. A hierarchical approach to concurrency control for multi databases. In Proceedings of the Second International Symposium on Databases in Parallel and Distributed Systems, 2002.

[12] Y. Breitbart and A. Silberschatz. Multi database update issues. In Proceedings of the International Conference on Management of Data, pages 135-142, June 2004.

[13] W. Du and A. Elmagarmid. Maintaining transaction consistency in HDDBSs using quasi serializable executions. Technical Report CSD-TR-969 Purdue University, March 2000.

[14] Y. Breitbart, W. Litwin, and A. Silberaschatz. Multi database concurrency control systems. Technical Report 154-89, Department of Computer Science, University of Kentucky, 2001.

[15] W. Litwin and H. Tirri. Flexible concurrency control using value dates. IEEE Distributed Processing Technical Committee Newsletter, 10(2):42-49, November 2002.

[16] C. Pu. Super databases for composition of heterogeneous data bases. In Proceedings of the International Conference on Data Engineering, pages 548-555, February 2003.

[17] D. Georgakopoulos and M. Rusinkiewicz. Transaction management in multi database systems. Technical Report

UH-CS-89-20, Department of Computer Science, University of Houston, September 2004.

Arun Kumar Yadav received the B.E. (Computer Science & Engineering) and M.Tech (Information Technology) degree in 2000 and 2004, respectively. Presently pursuing Doctorate Degree (Ph.D) in Computer Science from Singhania University, Rajasthan and working as Associate Professor & Head in the Department of Information Technology in Venkteshwar Institute of Technology, Indore (M.P.), India. His research interest includes Distributed Database Security, Data Structures and Algorithms. He is a member of IACSIT and IAENG.

Dr. Ajay Agarwal has done B.Tech. Degree in Computer Science & Engineering from Institute of Engineering & Technology, Lucknow (India), M.Tech.(honors) Degree in Computer Science & Engineering from Motilal Nehru Regional Engineering College, Allahabad and Ph.D. in Computer Science from Indian Institute of Technology, Delhi (India). Presently he is working as a Professor and Head in Computer Application Department at Krishna Institute of Engineering & Technology, Ghaziabad, India. He is a member of various Technical Societies viz. Institute of Electrical and Electronics Engineers (IEEE), Computer Society of India (CSI), Indian Society for Technical Education (ISTE), Institution of Engineers India, Institute of Chartered Computer Professional of India and Indian Association of Physics Teachers. He published many papers in various International/ National Journals and Conferences. His main research interests include: Distributed Database Security, Wireless Sensor Network, Mobile Computing and Middleware.

Kamlesh Chopra received the B.E. (Computer Science & Engineering) degree in 2006 and presently pursuing M.Tech (Computer Science & Engineering) from Rajiv Gandhi Technological University, Bhopal (M.P.). Presently working as Lecturer in the Department of Computer Science & Engineering in Venkteshwar Institute of Technology, Indore (M.P.), India. His research interest includes Distributed Database Security and Computer Networks.