


ular Concurrency Control and Failure Recovery, February 1988, pp. 146-159.


n. A Federated System Approach to Federated*


control protocol? In this paper we assumed that the GIM cannot take an advantage of knowledge about mixed types of local HIMs. For example, if one of the HIMs is rigorous and another one is strongly serializable, then the GIM assumes that each local HIM is strongly serializable; the knowledge that one of the HIMs is more restrictive (and, therefore, the GIM could be more permissive) is not used. Availability of such knowledge could possibly increase the current level of global transactions and improve transaction throughput.

Full data consistency and serializability can only be achieved in a way imposing restrictions that many consider severe. Thus, there is a need to find notions of consistency and ways of restricting “standard” notions can be stated rather than impossibility results. Other options for correctness include:

1. partitioned notions of consistency.

2. temporal consistency promises.

3. degree of consistency.
there will be a cycle in the approximate wait-for-graph. Clearly, the converse is not true; a cycle in the approximate wait-for-graph that is not a real deadlock is called a false deadlock. To reduce the likelihood of false deadlocks, the arc \( T_i \rightarrow T_j \) may be added to the approximate wait-for-graph only after \( T_i \) has been blocked for some threshold amount of time. These ideas are deadlock detection schemes of [8] and [61].

Very little work has been done to determine the performance of deadlock detection schemes. In particular, it will be important to evaluate the number of false alarms, and to compare detection schemes to simple timeouts. If some of the options we have reviewed are deemed not effective, then the deadlock

8 Conclusions

Multidatabases are one
committed. Note that aborting a transaction may imply that a compensating transaction needs to be scheduled at the sites on which compensatable subtransactions have successfully committed.

On receipt of the commit acknowledgment from the p-server, the GIM submits a commit to the remaining servers. If in case a subtransaction is aborted after the pivot has committed (not the subtransaction must be either a retrievable subtransaction or a redoable one), it or a redo transaction is executed for it depending upon its type.

The above protocol combines each of the schemes that we have discussed of global transactions. Obviously, we assume that each redo is appropriately restricted and serializability of assume that no other subtransaction of items read by each retrievable subtransac- able subtransactions. Recall there be no data depen-dion) global or w
conditions of the multilevel transaction model can be replaced by the isolation of recovery condition as we have discussed in our transaction model.

6.5 Combination of the Different Approaches

We have so far described the various approaches that have been studied in the literature in the context of the atomicity of global transactions in a multidatabase system. Each of the approaches has its merits and demerits. For example, while the redo technique seems attractive, it may not be suitable in all cases and depends upon the semantics of the transactions. Its applicability depends on the semantics of the applications. Further, their applicability is also limited. One interesting characteristic is that they can thus be classified into two categories:

...
The long transaction is broken up into subtransactions that commit and release their resources when completed. Long duration transactions are used for many scientific and engineering applications [36]. It is also shown that the log and state information needed for compensation can be stored within the same application database. The notion of sagas is extended into sagas, where a subtransaction may be further decomposed into steps that are ideas for using semantic atomicity for coping with long-lived activities.

One issue that we have not addressed in this section is the transaction. Note that some subtransaction may not say a subtransaction deposits funds in an account, may have been withdrawn by another transaction, customer a penalty or sending a message.

Not be compensatable, e.g., financial discussion in the literature.

Compensation for multi...
Two-phase commit (2PC) protocol is introduced to guarantee semantic atomicity. The protocol works as follows.

When a transaction completes, the GIM sends "prepare" messages to the servers at each site, as it is done in the 2PC protocol. However, unlike the 2PC protocol, upon receiving the "prepare" message, the servers optimistically try to commit their subtransactions at that point. The result is reported to the GIM. If all subtransactions committed, then the transaction is committed. If not, the transaction is declared aborted, and compensating transactions are required. In the common case where subtransactions at the 2PC site sites commit sooner than in the 2PC protocol, the 2PC protocol was also developed independently in [45].

Without an atomic global commit protocol is also subject to conflicts, commit protocols do not require each local multi-site transactions and are thus attractive. What have so far ignored the fact that others may violate database constraints. If transactions $T_1^i$ and $T_2^i$ execute, it is possible...
reservation since the flight is already full. Had $T_1$ not executed $T_2$ would have been able to procure the reservation. Thus, the state that results after the execution of $CT_1$ differs from the state that would have resulted had $T_1$ not executed at all. This, as in the current flight reservation, is nevertheless quite acceptable.

We stress that compensating transaction for a committed global subtransaction is:

regular transaction and, thus, it must preserve database consistency only consist of an inverse function of the original subtransaction and other actions. In our example, for instance, the bank could have triggered another transaction (reflecting that the flight is cancelled) and submitted the corresponding.
subtransaction, since those reads are now invalid. In other words, there are no data dependencies between $T_i^2$ and any other subtransaction of $T_i$. Techniques such as [41, 35] can be used for checkpointing transaction programs and tracking data dependencies among subtransactions.

Further, it must be the case that subtransaction $T_i^2$ is retried [51]; that is, if $T_i^2$ is retried, the state of the local DBM execution of other local transactions. This should not result in the database state. It must be noted that not every subtransaction can be committed. It must be noted that not every subtransaction can be committed.

for example, a subtransaction that is to be for example, a subtransaction that is to be retried, depending upon the balance in the account. if a subtransaction is to be retried, depending upon the balance in the account. if it is retried a sufficient number of times (from any database state) it will eventually commit. The technique for example, a subtransaction that is to be retried, depending upon the balance in the account. if a subtransaction is to be retried, depending upon the balance in the account.
Another option of ensuring global serializability is to use some mechanism for preventing cycles in the global serialization graph through indirect conflicts between global transactions. Note as discussed in Section 4, executing global transactions serially, or using one of altruistic locking or the commit graph approach can be used for this purpose. in [10, 11] uses the commit graph approach to prevent cycles through it is assumed there that local HRs follow the strict 2PL schedules) and rigorousness of GS is ensured (by naive 2PL locking scheme on global locks) to ensure. Walski and Väjänen also propose however, requires this equivalently
of failures. If we were to ensure the 2LSR correctness criterion of global schedules, then besides ensuring
serializability, we must further ensure that the projection of the global schedule
operations belonging to global transactions (which we refer to as $G_s$) is also serializable.
that to ensure serializability itself, the GIM needs to ensure rigorous
the first more restrictive restriction on the interactions between global
 every rigorous schedule is also serializable, 2LSR
were to adopt the more restrictive second restriction that the schedule $G_s$ is
the global schedules are 2LSR because depending upon the possible
transaction $T_3$ is executed to redo the write operations performed by the globally committed but locally aborted transaction $T_1$. In that example, since the local DBS considered $T_3$ as a different transaction than $T_1$, the resulting local schedule was not serializable from the GIM view.

Note that each of our correctness criteria discussed in Section 4 and 5 (that is,global 1SR, or 2ISR) requires that the schedules at the local DBS be serializable from the MBS point of view. We refer to the local schedule as being serializable if the MBS point of view. M-serializability can be defined as

**Definition 6.1:** Let $S_j$ be a local schedule of transactions and redo transactions. Let $n$ be also over the read operations performed by $S_j$ but are committed by $T_i$ and the write operations of $T_i$ is considered serializable.

If the global
sites, regarding issues such as error handling and who controls the global commit. If there were a single standard 2PC protocol, these problems would be avoided, but it is unlikely that this will occur. Already there are several competing “standards” (e.g., LU-2 [15], OSI TP [63]).

problem of coordinating heterogeneous commit protocols will persist. Some initial coordination is reported in [37].

As we argued in Section 3, there may be cases where the prepare-provided by all sites. This may be due to the following:

1. Sites only offer a Service Request interface, giving control over service commit;

2. Sites wish to retain their execution or coordination;

3. Performance of 2PC in a distributed due to remain in the prepared state time and throughput remain.

In the rest of this section, as being used.

6.2 Redo A:

Consider the following: In this s...
3. Compensate. A each site where a subtransaction of a global transaction did commit, a compensating subtransaction is run to semantically undo the effects of the committed subtransaction.

We discuss these approaches in Sections 6.2 through 6.4. While redo and retry techniques preserve the standard atomicity of transactions, in the case of compensation a weaker notion is used, since it is possible that the effects of the aborted global transaction are not preserved. This impacts the preservation of consistency in transactions. This impacts the preservation of consistency in transactions. This impacts the preservation of consistency in transactions. Finally, each of the above techniques can be used to combine them into a single uniform solution.

6.1 Two Phase Commit

If the local DBMSs support a pre-defined control mechanism, the execution of the executables of each
A different notion of correctness is used in [27]. Here transactions are grouped into disjoint types. An application administrator then determines that transactions of certain types can be interleaved arbitrarily without causing constraints to be violated. For example, in a bank it be safe for deposit transaction to interleave with other deposits and withdrawals.

The concurrency control mechanism proposed in [27] uses local locks and global locks to avoid undesirable interleavings.

The concept of compatibility is refined in [44] and states that actions are defined. These levels are structured to include those at lower levels. Further actions which represent points in the use of compatible actions.
In Example 5.3, we could say that there is a second type of correctness criteria, in addition to strong correctness. In this case we do not want the transfer transaction to be involved in a serialization cycle. One "artificial" way of dealing with this problem is to declare item \(tot al\) and define an integrity constraint \(tot al = a + b\). If this constraint schedule of Example 5.3 would not be strongly correct and would be

However, one could argue that defining additional constraints be no real integrity constraint between accounts \(a\) and \(b\) not equal \(tot al\). That may be special. If we declare the constraint between audits. Second, if we constraints between accounts

gener
serializable. A different idea for enforcing global constraints is presented in [5]. The claim is that global constraints tend to be very simple in practice and that the GIM can enforce them directly without concerning itself with serializability. A second claim is that global constraints can be "approximate," giving the GIM even more flexibility in enforcing them.

To illustrate, consider a copy constraint between item $g_1$ at site $s_1$, especially if they run on independent sites, can tolerate constraints of the form $|g_1 - g_2| \leq \epsilon$, where $\epsilon$ is some small number. Every update to $g_1$ needs to be reproduced and can keep track of a window of allowable values. The new values are not propagated, as apparent when failures are not satisfied. In summary, in [5],
1. If global transactions are not allowed to access local data, then we can drop the GPR requirement. (Actually, if global transactions cannot read local data, then they are necessarily GPR). So the requirement is not dropped; it is replaced by a more restrictive one. If we assume that local transactions cannot read global data, and that the global transactions are not allowed to read local data, then the local and global data is totally decoupled, and always be serializable, without any requirements on the transactions.

2. If there are no global/local constraints, then the proof of this is lengthy [48], but the intuition is that local schedules at each site are serializable (regardless of whether they are executed by global transactions or not).
Definition 5.3 [49]: A global schedule $S$ is two-level serializable (2LSR) if it is LSR and its projection to a set of global transactions is serializable.

Globally serializable schedules are always 2LSR, but the converse is not true. This is illustrated by the following example, which also shows that 2LSR schedules may violate constraints that contain "unusual" transactions:

Example 5.2 [49]: Consider an MIS where there is a single local global item, $b$ and $c$ at $s_1$ and $d$ at $s_2$. There is one global:

$$a > 0 \rightarrow b > 0$$
$$d > 0 \rightarrow (b > 0 \text{ or } c > 0)$$

Consider the following two global and one local transactions:

$T_1$: if $(a \leq 0)$ then
  $d := 1$

$T_2$: if $(a > 0)$ then
  $c := 0$

$L_3$:

Starting from a state where all

$S_1$

The resulting schedule is:
autonomy. Note that transactions that only contain assignment and alternation statements can always be converted into fixed structure transaction. For example, as we illustrated earlier, the transactions in Example 5.1 could be made fixed structure. However, if transactions contain there may not be an easy way to make them fixed structure.

In [17] a third strategy has been suggested for making ISR schedules preserve a transaction $T_i$ is ND if it has no write dependencies [17]; that is, if its actions depend in any way on the values read at another site. Both $T_1$ and $T_2$ in need dependencies. If a transaction is ND then it is clearly LIP. The concept of a transaction that writes into item $\theta$ a value read elsewhere, but as noted in [17] this is more restrictive.

5.2 Two Level Serializability

The notion of local serializability has two levels: (a) a serializable schedule where:

1. Isolates
   1.1. Consistency
   1.2. Isolated

2. Isolates
   2.1. Consistency
   2.2. Isolated
We use the notation \( r_i(a, x) \) (or \( w_i(a, x) \)) for a read (write) action of transaction \( T_i \) on item \( a \), where \( x \) is the value read (written).

\[
S_1: \ w_i(a, -1) \ r_2(a, -1) \\
S_2: \ w_2(b, -1) \ r_1(b, -1)
\]

Each local schedule is serializable. Nevertheless, the final state \( a = -1, b = -1 \) is invalid.

One simple way to avoid the type of problems shown in Example 5.1 above is to make local sites run a two-phase locking (2PL) protocol, and, in addition, to follow the 2PL protocol in acquiring and releasing of local locks. Thus, local transaction \( T_2 \) has read the value of \( b \) (since it may read and \( T_2 \) would not release the lock) and \( T_2 \) would not release the lock until \( T_1 \) releases the lock. A deadlock may result.
5 Alternative Consistency Notions

As we have seen, guaranteeing global serializability may result (in some environments) in poor performance due either to a low degree of concurrency or the large number of aborted transactions.

Moreover, as we shall see later, when we discuss failures, it is very hard to obtain global serializability in some cases. Thus, several researchers have suggested notions of weaker than global serializability. In this section we survey some that neither failures nor unilateral aborts of global transactions.

5.1 Local Serializability

Global serializability guarantees that neither failures nor unilateral aborts of global transactions.
Example 4.1: Consider a multi database system located at two sites: \( s_1 \) with data items \( a \) and \( b \), and \( s_2 \) with data items \( c \) and \( d \). Let \( T_1 \) and \( T_2 \) be two read-only global transactions, and let \( T_3 \) and \( T_4 \) be two local transactions. The schedules at sites \( s_1 \) and \( s_2 \) are, respectively:

\[
S_1 : w_1(a) c_1 r_3(a) w_3(b) c_3 r_2(b) c_2
\]

\[
S_2 : w_2(x) c_2 r_4(x) w_4(y) c_4 r_1(y) c_1
\]

At each site, the schedules can be produced by strict two phase locking at each site. However, the dependencies \( T_1 \rightarrow T_3 \rightarrow T_2 \) and \( T_1 \rightarrow T_4 \rightarrow T_2 \) are not serializable.

This problem can be avoided if the GIM does not issue any of its actions have been completed. In Example 4.1, if \( r_1(y) \) at \( s_2 \) is acknowledged. In turn, \( T_3 \) could not take place. It may lead to the commit of a transaction until a transaction commits until a transaction arises.
To see why the knowledge that local sites generate strongly recoverable schedules (as opposed to strongly serializable or sp-schedules) leads to higher concurrency of the global concurrency control mechanism let us return to the strategy of executing global transactions serially at sites that generate strongly serializable or sp-schedules, the GIMavoids cycle of transactions do not overlap (see Section 4.2.1). With strongly recoverable schedules, it is sufficient to ensure that transactions do their computing in a global transaction issues its first commit before any other global transaction issues any other commit. This is an example of a modified version of Example 4.4.

The actions in this example (i.e.,...
Since local sp-schedules are strongly serializable, it is possible to use the global concurrency control schemes outlined in the previous section. However, if each local IMS notifies the central site in advance what action will constitute the serialization point, then one could obtain serializability more efficiently. For example, a timestamp scheduler might indicate that the time the transaction was submitted is the serialization point (i.e., when the transaction runs). In the general model, each site could define a different action to indicate what actions are serialization points. One site could say first actions are serialization points, another site could say last actions are. The global concurrency control system [46]. As before, the key issue is that cycles are executed locally.
at a site. However, the lock is not fully released; it is left in a "marked" state. Other transactions that request a site lock that is marked, can obtain the lock, but are then forced to be in the wake of the original transaction. The GIM must ensure that the relationship "is in the wake of" cycles. The latter can be done by keeping a *site-graph* in which there is an edge between $T_i$ if $T_j$ is in the wake of $T_i$.

All the mechanisms we have described for strongly serializable local schedules.

However, optimistic versions can easily be developed. For instance, instead of delaying transactions involved in a cycle, we could...

4.2.2 Serialization-Point Based

The notion of a strongly serializable schedule of transactions each of which is a distinct
Definition 4.1: Let $S$ be a serializable schedule. We say that schedule $S$ is strongly serializable if and only if for every two transactions $T_i$ and $T_j$ in $S$, if the last operation of $T_i$ (commit or abort) precedes the first operation of $T_j$, then there is some serial schedule equivalent to $S$ that precedes $T_j$ (i.e., $T_i$ precedes $T_j$ in the $S$'s serialization order). □

Assuming that a transaction receives a timestamp at the time of execution, the basic timestamp ordering concurrency control algorithm ensures serializable. Thus, as shown above, the GIM can ensure strongly serializable by executing global in which we can do better. For example, no need to execute themselves (the locks are kept at) must acquire it.
increment it and write an incremented value into the database. Thus, the ticket value read indicates the serialization order of the global transactions at the site.

The algorithm of [33] is optimistic: the GIM keeps a serialization graph for all actions (started but not committed). When a transaction $T$ reads ticket value $t$ at $s_i$, it enters from every transaction that read a ticket less than $t$ at $s_i$ to actions and is not involved in a cycle, it is committed, else the ticket method guarantees global serializability.

The ticket idea can also be used in a distributed system where transactions are assigned a priori a global serial number advance. If a transaction aborts, the per-transaction serial number can be increased.
To avoid these cycles, the GIM will have to take some action. What action is taken depends on the amount of knowledge the GIM has concerning the local concurrency control mechanisms.

In the subsections that follow we consider various scenarios, and for each one explain the types of GIM actions that will ensure global serializability. The base scenario (Section 4.1) corresponds to our base transaction model (Section 2): the GIM simply knows that each local transaction is local serializable and deadlock-free schedules. Thus, the GIM considers the sites (black boxes).

In general terms, the actions taken by the GIM can be of two types:

- **Pessimistic.** Global transactions are delayed to avoid cycles or potential cycles.

- **Optimistic.** Cycles or potential cycles are detected and resolved.

There is a tradeoff between these two. Optimistic action aborts but may result in a deadlock or concurrency but may require more information.

4.1 Integration

(Cont)
they claim the operator at a site can always manually release a transaction that hangs for too long (e.g., break locks manually). So if a transaction ever waits too long in its prepare-to-commit state, it can be aborted. The second camp then argues that if the prepare-to-commit can be broken by the operator, then sites can unilaterally abort after all, so we have another.

Without taking sides in the argument, we believe it is important to study or without prepare-to-commit at the sites. In this paper, we will

3.3 Global Deadlock Problem

Consider a multivatabase system where each local serializability. We assume that each local DB
deadlocks. However, in such systems the detected by the GIM

Example 3.3: Consider an
b, and s2 with data item
to guarantee local
provide a prepared state for each subtransaction. The subtransaction should remain in the prepared state until the coordinator decides whether to commit or abort the transaction.

If we wish to preserve the execution autonomy of each of the participating local IMSs, then we must assume that local IMSs do not export a transaction's prepared state. In such an environment, a IMS can unilaterally abort a subtransaction any time before its completion; not only leads to global transactions that are not atomic, but to incorrect global results, as illustrated below.

**Example 3.2:** Consider a global database consisting of two sites with data item c. Consider the following global transaction:

$$T_1: r_1(a) w_1(a)$$

Suppose that $T_1$ has completed its read/write actions and requests to both sites. Site $s_2$ receives the commit requests and $s_1$ decides to abort its subtransaction. The IMS undoes the $T_1$ actions. The global transaction is executed and completed.

At this point, the correct state of the global database would be:

$$s_1: a = x$$
$$s_2: a = y$$

where $x$ and $y$ are the original values of the data item $c$. The IMSs must ensure that all data items are in a consistent state, regardless of whether the transaction is committed or aborted. This is achieved through the use of transaction isolation levels, such asSerializable isolation, which guarantees that transactions are isolated from each other and from the system's concurrency control mechanism. This helps prevent anomalies, such as dirty reads and phantom reads, which can occur when multiple transactions access or modify the same data item at the same time.

Furthermore, the use of a global transaction manager (GTM) can help manage the coordination of local transactions and ensure that the global database remains consistent. The GTM is responsible for ensuring that all transactions are executed in a coordinated manner, taking into account the need to maintain consistency, isolation, and durability of the global database. This is achieved through the use of distributed transaction protocols, such as 2PC (two-phase commit), which ensures that all transactions are either committed or rolled back in a coordinated manner across all participating sites. This helps prevent anomalies, such as theacosyphon anomaly, which can occur when a transaction is committed on one site but not on another, leading to a different state of the global database.
about the concurrency control mechanisms used at local sites. For each extension, we will study how global transaction management is affected.

3 Multidatabase Transaction Management Issues

The Global Transaction Manager (GTM) should guarantee the ACID properties of global transactions, even in the presence of local transactions that the GM is not aware of. In addition, the GM should guarantee deadlock-free executions of global transactions and means to recover from any type of system failures. In the next three subsections, difficulties that may arise.

3.1 Global Serializability Problem

The various local DBMSs may use different concurrency control schemes (2PL, Timestamp Ordering, Seriation Graph Testing, SGI, etc.). Ensuring global serializability in a homogeneous distributed database system that uses the same concurrency control scheme and shares its control information cannot be used in a MIS environment.

Since local transactions execute outside the control of the GTS, serializability only through the control of the GTS is not possible in such an environment, even a shared serializability. The following example illustrates.

Example 3.4

$b, c$
operations submitted to it. We do assume that the actions of a given transaction at a site always end an execution with a commit (or abort) operation.

We do not impose any restrictions on how the various read and write operations of a global transaction are executed by the GIM. It is possible in our model for several operations of the transaction to be executed by the GIM at the same time (parallel execution) or for an operation of the transaction (except the very first one) to be submitted for execution until an acknowledgment from the previous operation of the same transaction.

As mentioned earlier, we will also consider variations of this overall multibased system model. In particular, we will consider two variations:

- **Service Interface.** Many real life high-level service interfaces interbank clear request
- **local transactions**, those transactions that access data managed by only a single **MIS**. These transactions are executed by the local **MIS**, outside of **MIS** control.

- **global transactions**, those transactions that are executed under **MIS** control. A transaction consists of a number of subtransactions, each of which is seen as a transaction from the point of view of local **MIS** where the subtransactions are executed.

  The local schedule at site \( s_k \), denoted by \( S_k \), is a sequence of operations resulting from their execution at site \( s_k \). If \( S_k \) contains \( c_i \) \((a_i)\) operation, \( T_i \) is aborted in \( S_k \). A projection of \( S_k \) on a set of operations of transactions from \( S_k \) contains only operations from \( \{a_i\} \). We say that transaction \( T_i \) contains operations from \( S_k \) if \( T_i \) contains operations from \( \{a_i\} \) and \( T_i \) does not contain operations from any other transactions except \( \{a_i\} \).
- **Get-serialization-order.** Retrieve information regarding the commit order of transactions. (Such an order can be represented by a serialization graph, where vertices correspond to transaction names, and the set of edges specifies an acyclic in the serialization graph indicates a non-serializable transaction.)

- **Inquire.** Find out status (e.g., commit, abort)

- **Disable transaction class.** If a transaction class type, or by read or write

Thus, each local database invocation by users to the transaction uses...
may result in performance degradation, and, further, may render pre-existing applications inoperative.

- **Execution autonomy:** Each local DBMS should retain complete control over the execution of transactions at its site. An implication of this constraint is that a local transaction executing at its site at any time during its execution, is not subject to visibility or control by a global transaction in the process of being committed by the system.

- **Communication autonomy:** Local DBMSs integrate and coordinate the actions of global transactions executing on their behalf. Local DBMSs do not share their control structures with each other. Participating DBMSs may have different autonomy levels and may not have the same level of communication capability. The way to characterize the autonomy is not just in terms of communication but also in terms of how the user interface and the user's ability to interact with the system are designed.

One way to characterize the autonomy levels is to consider the user interface. The user can interact with the system through an external user interface or a user interface embedded in the transaction manager. External users transact through the user interface provided by the DBMS, while internal users transact directly with the transaction manager.
1 Introduction

Recent progress in communication and database technologies has changed the user data processing environment. The present data processing situation is characterized by a growing number of applications that require access to various pre-existing local data sources located in heterogeneous hardware and software environments distributed among the nodes of a network. Each local data source is a collection of data and applications that are run under a particular database management system (DBS) and are administered/operated under a particular policy or local management. The data sources are pre-existing in the sense that they were created in an uncoordinated way and without considering that one day they may need to cooperate with others.

The DBSs involved are heterogeneous in the sense that they operate in different environments and use different underlying data models, data definition and data manipulation languages and concurrency control mechanisms, and physical data organization. A database is composed of local data sources. Systems that provide access methods to a single database system and hide from users the location or the characteristics of the database access methods. It provides users with uniform access without migrating the data to a new location or the characteristics of the database access methods.

A multi database approach is different from the single database approach system. Each database has its own management and control mechanisms, and the user has access to the data in each database.
Overview of Multi database Transaction Management

Yuri Breitbart\textsuperscript{1}
Department of Computer Science
University of Kentucky
Lexington, KY 40506

Hector Garcia-Molina
Department of Computer Science
Stanford University
Stanford, CA 94305

Ali Silberschatz\textsuperscript{2}
Department of Computer Sciences
University of Texas
Austin, TX 78712

Abstract

A multi database system (MBS) is a facility that allows users access to data located in multiple autonomous systems. In such a system, global transactions are executed under the control of the local DBSs. Each MBS participates in a global transaction. In addition,