


A. P. Sallent: A transaction approach to federated databases.


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

Full data consistency and serializability can only be achieved in a non-imposing restrictions that many consider severe. Thus, there is a need for new 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 correctness.
2. temporal consistency promises (....)
3. degree
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 lead to 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 detections produced by these schemes, and to compare detection schemes to simple timeout methods. It is clear that some of the options we have reviewed are desirable in certain situations, and strategies where there is no global timeout are not effective, then the deadlock detection.

8 Conclusions

Multi-databases are one of the major foundation (USA) multi-database researchers and sponsors. They have sponsored research that multi-
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 pivot 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 scheme that we have discussed of global transactions. Obviously, we assume that each redo is appropriately restricted and m serializability of assume that no other subtransaction of items read by each retrievable subtransasctions. Recall there be no data depen tion) global or wo
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 on the atomicity of global transactions in a multidatabase system. Each of the approaches has merits and demerits. For example, while the redo technique seems at first glance to offer the same degree of isolation as the commit technique, it is less applicable in practice. Each approach has its own semantics, and these semantics are dependent on the characteristics of the transactions, the data items accessed, and the system model.

One interesting characteristic of the redo technique is that it can be used in a variety of situations, and it can thus be applied to a wide range of problems.
system. 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 be stored within the same application database. The notion of sagas is extended in sagas, where a subtransaction may be further decomposed into steps that are ideas for using semantic atonality for coping with long-lived activities.

One issue that we have not addressed in this section is 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. It not be compensatable, e.g., firing discussed 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 rolled back to undo all the subtransactions that did not commit. In the common case where subtransactions at multiple sites commit sooner than in the 2PC protocol, each 2PC site can commit sooner.

The 2PC protocol was also developed independently in [45], with an atomic global commit protocol also supporting multi-site transactions and, thus, atomicity.

We have so far ignored the fact that others may violate database constraints. If $T_1^i$ and $T_2^i$ execute simultaneously, it is possible that...
reservation since the flight is already full. Had $T_1^i$ not executed $T_2$ would have been able to procure the reservation. Thus, the state that results after the execution of $CT_1^i$ differs from the state that would have resulted had $T_1^i$ 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 a 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 could have triggered another transaction (reflecting that the flight is states that if the the co...
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; that is, if \( T_i^2 \)
sufficient number of times (from many database state) it will eventually commi-

since before the subtransaction is retried the state of the local DBM
exe cution of other local transactions. This should not result in

cannot be committed. It must be noted that not every time

for example, a subtransaction that is to be re-

retried, depending upon the balance in

hand, if a subtransaction is to

if it is retried a sufficient

The technique

at o ni city of
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 TRMS follow the strict 2PL schedules and rigorousness of GS is ensured (by mal 2PL locking scheme on global locks) to ensure Wijski and Väijäläinen also propose and it assumes that the path however, requires the equivalent
of failures. If we were to ensure the 2LSR correctness criterion of global schedules, then besides ensuring m 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.

In order to ensure m serializability itself, the GIM needs to ensure rigorous schedules. The first more weaker restriction on the interaction between global transactions is that every rigorous schedule is also m serializable. 2LSR imposes a second more restrictive requirement: the global schedules are 2LSR if and only if the schedule \( G_S \) is m serializable, 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 HB$S$ considered $T_3$ as a different
transaction than $T_1$, the resulting local schedule was not serializable from the GIMview.

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 HB$S$s be serializable from
the MBS point of view. We refer to the local schedule as being \textit{serializable} if
serializability can be defined as

\textbf{Definition 6.1:} Let $S_j$ be a local schedule consisting of actions and redo transactions. Let $m_j$ be all operations performed by
HB$S$ but are committed by $T_j$ and the write operations of $T_j$ is considered
serializable.

If the global

\[ \text{...} \]
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 commitment;

2. Sites wish to retain their execution or control;

3. Performance of 2PC in a distributed system may require time and throughput requirements.

In the rest of this section, we will be discussing the

6.2 Redo Approach

Consider the following scenario:

In this scenario,
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 are 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 transactions. This impacts the preservation of consistency in issue in Section 6.4. Finally, each of the above techniques to combine them into a single uniform solution.

6.1 **Two Phase Commit**

If the local DBMSs support a pre-consistency in a failure control mechanism, the solution [7] (or one of its execution to 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, it might be safe for deposit transactions to interleave with other deposits and with 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 several levels of actions are defined. These levels are structured so that actions can include those at lower levels. Further constraints are enforced by 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 totals and define an integrity constraint \( t_{otal} = a + b \). 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 \( t_{otal} \), that may are special. If we declare the constraint audits. Second, if we constraints bet-
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 $s_1$ and its replica at site $s_2$. Every update to $g_1$ needs to be reproduced at $s_2$. The constraint may be $|g_1 - g_2| \leq \epsilon$, where $\epsilon$ is some threshold. The GIM can keep track of a window of allowable deviations in the values of the new values are not properly synchronized at the same site. In summary, in [5] it is claimed that serializability is satisfied by the GIM.
1. If global transactions are not allowed to access local data, then we can drop the GF requirement. (Actually, if global transactions cannot read local data, then they are necessarily GF. 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 cannot write local data, then the local and global data is totally decoupled and always be serializable, without any requirements on the transaction scheduling.

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

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

**Example 5.2 [49]:** Consider an MTS where there is a single local global item $x$, $b$ and $c$ at $s_1$ and $d$ at $s_2$. There is one global transaction $T_1$:

$$a > 0 \rightarrow b > 0$$

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

Consider the following two global and one local transaction:

$$T_1: \quad \text{if } (a \leq 0) \text{ then}$$

$$d := 1$$

$$T_2: \quad \text{if } (a > 0) \text{ then}$$

$$c$$

$$L_3:$$

Starting from a state where all

$$S_1$$

The resulting schedule
autonomy. Note that transactions that only contain assignment and alternation statements can always be converted into fixed structure transactions. 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 preserved: a transaction $T_i$ is ND if it has no value 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 transaction $T_1$ contain dependencies. If a transaction is ND then it is clearly LIP. The converse is not true; transaction $T_1$ writes into item $b$ a value read elsewhere, but as noted in [17] this is more restrictive.

5.2 Two Level Serializability

The notion of local serializability: local data and

1. local. Consists

2.
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).

\[
S_1: w_1(a, -1) \quad r_2(a, -1) \\
S_2: w_2(b, -1) \quad r_1(b, -1)
\]

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

One simple way to avoid the type of problems shown in Example 5.1 above is for the local sites to run a two-phase locking (2PL) protocol, and, in addition, the 2PL policy in acquiring and releasing of their local locks. If the 2PL protocol is followed, then transaction \( T_1 \) has read the value of \( b \) (since it may read \( b \) before \( T_2 \) reads it) and \( T_2 \) would not release the lock until \( T_1 \) has released the lock. This avoids the deadlock.
5 Alternative Consistency Notions

As we have seen, guaranteeing global serializability may result 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 it is to be dropped, usually
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 $T_2$ schedule is not serializable.

This problem can be avoided if the GIM does not insist that all of its actions have been completed. In Example 4, $r_1(y)$ at $s_2$ is acknowledged. In turn, $T_3$ could not take place. (It may lead to blocks until a transaction commits until a transaction commits serializability. No
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. Sites generate strongly serializable or sp-schedules, the GIMavoid cycles transactions do not overlap (see Section 4.2.1). With strongly recoverable global transactions it is sufficient to ensure that transactions do their console a global transaction issues its first console statement before any other global transaction issues any console statements. A modified version of Example

The actions in both examples (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 transaction submitted is the serialization point (i.e., when the transaction runs). In general, each site could define a different action to be the serialization point. One site could say first actions are serialization points and another site could say last actions are serialization points. The global concurrency control [46] is designed to handle such cases. As before, the key issues are execution order and support for

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 *wake-graph* in which there is an edge between $T_j$ if $T_j$ is in the wake of $T_i$.

All the mechanisms we have described for strongly serializable local schedules lead to a strong serializable global schedule. However, optimistic versions can easily be developed. For instance, instead of delaying the transaction involved in a cycle, we can...

### 4.2.2 Serialization-Point Based

The notion of a strongly serializable global 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 $S$ which 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 serializability. 3 Thus, as shown above, the GMT can ensure strongly serializable by executing global in which we can do better. For example, no need to execute them excessively (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 GM keeps a serialization graph for all actions (started but not committed). When a transaction \( T \) reads ticket value \( t \) and entered from every transaction that read a ticket less than \( t \) at \( s_i \), the ticket method guarantees global serializability.

The ticket idea can also be used in a global algorithm. The per-transaction

abort logic.
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 each site a "black box." In subsequent scenarios, the GIM assumes additional knowledge about 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.
- **Optimistic.** Cycles or potential cycles are resolved.

There is a tradeoff between these two types of action aborts but may result in lost concurrency but may result in lost data.

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 counter-argues that if the prepare-to-commit condition can be broken by the operator, then sites can unilaterally abort after all, so we might as well do one.

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 study

3.3 Global Deadlock Problem

Consider a multi database 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 DBS$s$, then we must assume that local DBS$s$ do not export a transaction's prepared state. In such an environment, a DBS can unilaterally abort a subtransaction any time before its completion. This not only leads to global transactions that are not atomic, but to incorrect global states, as illustrated below.

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

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

Suppose that $T_1$ has completed its read/write actions and requests to both sites. Site $s_2$ receives the completion messages first and decides to abort its subtransaction. Note: the DBS$S$ undoes the $T_1$ actions.

is executed and committed.

At this point, the global database is in a correct state.
about the concurrency control mechanisms used at local sites. For each extension, we will study how global transaction management is affected.

3 Multi-database 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 GTMs is not aware of. In addition, the GTMs should guarantee deadlock-free executions of global transactions and it should 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 (TO), Sridharan's Graph Testing (SGT), etc.). ensuring global serializability in a homogeneous distributed database the same concurrency control scheme and shares its control cannot be used in a MDS environment.

Since local transactions execute outside the control of the serializability only through the control of the serializability in such an environment, even a serializability. The following

Example 3.1

\[ b, a, c, d \] for
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 same transaction to be executed by the GIM at the same time (parallel execution) or for each of the transaction (except the very first one) to be submitted for execution until an acknowledgment from the previous operation of the same transaction is received. The overall multi-database system model is depicted in Figure 1.

As mentioned earlier, we will also consider variations in particular, we will consider two variations:

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

- **global transactions**, those transactions that are executed under MIS control and consist of a number of subtransactions, each of which is a transaction from the point of view of local IMS where the subtransaction aborts.

  The local schedule at site $s_k$, denoted by $S_k$, is a sequence of operations resulting from the execution at site $s_k$ if $S_k$ contains $c_i(a_i)$ operation. Transaction $T_k$ aborts in $S_k$. A projection of $S_k$ on a subset of operations of transactions from $S_k$ contains only operations of $T_i$. We say that transaction $T_i$ is contained in $S_k$ and $T_i$ does $c_o$.
- **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 specify an acyclic in the serialization graph indicates a non-serializable transaction.)

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

- **Disable transaction class.**

Thus, each local data item invoked by users to the transaction uses...

...
may result in performance degradation, and, further, may render pre-existing applications inoperative.

- **Execution autonomy**: Each local DBS 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 a global transaction in the process of being committed by the

- **Communication autonomy**: Local DBSs integrate and coordinate the actions of global transactions executing in other systems. Local DBSs do not share their control system.

Participating DBSs may have different autonomy to participate in the coordination of others, or not. High communication among other source offers to user transactions external users transactional users transactions; they access.

si
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 (DBMS) and are administered/operated under a particular policy or local constraints.

The data sources are pre-existing in the sense that they were created independently and without coordination among themselves. They may need to be updated or modified without considering that one day they may need to be linked to others.

DBMSs 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 structures.

A database is composed of local data sources. Systems that have many different local data sources are called heterogeneous systems. Logically, a single database system hides from users the use of access methods. It provides users with uniform access without migrating the data to a new location or the characteristics of the various databases involved.

A multiaccess database approach is needed to access data in various locations with different local access.
Overview of Multi-database Transaction Management

Yuri Breitbart
Department of Computer Science
University of Kentucky
Lexington, KY 40506

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

Ali Silberschatz
Department of Computer Sciences
University of Texas
Austin, TX 78712

Abstract

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