Maintaining Database Consistency in Presence of Value Dependencies in Multidatabase Systems

Claire Morpain — LSI Laboratory, e-mail: morpain@lsi.fr
Michèle Cart — LSI Laboratory, e-mail: cart@lsi.fr
Jean Ferrié — LSI Laboratory, e-mail: ferrie@lsi.fr
Jean-François Pons — LSI Laboratory, e-mail: pons@lsi.fr

Abstract

The emergence of new criteria specifically adapted to multidatabase systems, in response to constraints imposed by global serializability, leads to restrictive hypotheses in order to ensure correctness of executions. This is the case with the *two level serializability* presented in [6], that ensures *strongly correct* executions if transaction programs are *Local Database Preserving (LDP)*. The main drawback of the LDP hypothesis is that it relies on rigorous programming. The principal objective of this paper has been to suppress this drawback while conserving the strong correctness of 2LSR executions We propose defining precisely the notion of value dependencies, and managing them so as not to impose the LDP property.

1. Context of multidatabase systems

A multidatabase system is built from a set of independent and heterogeneous database systems (sites), in a manner that allows users to access objects residing at different sites. Each site has a database management system which manages a set of objects linked together by integrity constraints -the local database. These objects are divided in two sub-sets: the set of local data items (LD) and the set of global data items (GD). A local object can be linked with any objects at the same site, either local or global. A global object can be linked with any objects at the same site (local or global) and also with global objects at other sites.

Consequently, three types of integrity constraints are distinguished according to the nature of involved objects: local integrity constraints (LIC) between local data items (necessarily located at the same site), global integrity constraints (GIC) between global data items (possibly located at the same site), and integrity constraints between local and global data items (necessarily located at the same site). Replicated objects are therefore defined as global objects because a GIC must be defined in order to maintain the same value for these objects at different sites. In that context, union of local databases is referred as the global database. A state of the database corresponds to the mapping of a value to each object and it is said consistent if all object values agree with integrity constraints.

Objects are accessed by transactions resulting from executions of transaction programs. A transaction program [7] is written in a high level language. Its execution results in read or write operations performed by a transaction according to the R/W model. In a multidatabase system there are two kinds of transactions: local transactions (LT) resulting from executions of local transaction programs (L) and accessing objects at only one site, and global transactions (GT) resulting from executions of global transaction programs (G) and accessing objects at several sites. Each global transaction GT_i is decomposed into global subtransactions (GST), one per site. GST_{ik} is the set of GT_i's operations executed at site k. In addition, it is possible for a global transaction program G_i to have value dependencies. A value dependency (VD) expresses that the value given to an object located at one site possibly depends on the value of an object located at another site.

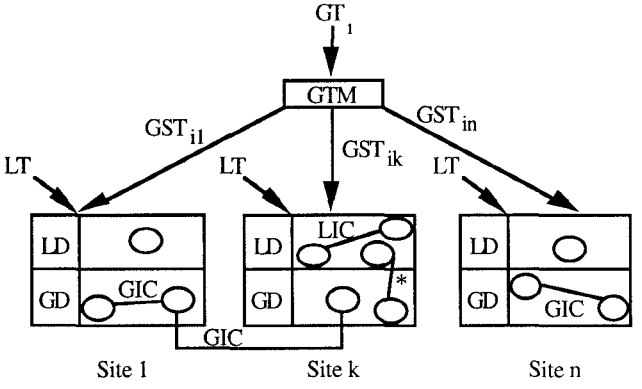


Figure 1. Architecture of a multidatabase system

Notation: —*— Integrity constraint between local and global objects.

In that context, we suppose that each site guarantees, by means of a local transaction manager, serializability and fault tolerance of the local transactions and the global subtransactions accessing the objects of the site. It thus preserves the ACID properties of global subtransactions.

As for GTs, they are controlled by a Global Transaction Manager (GTM). This control must be done while

* This research was supported in part by ARCTICA project n° 95S0143, coordinated by the Ministère de l'Education Nationale, de l'Enseignement Supérieur et de la Recherche.

SIGMOD '96 6/96 Montreal, Canada
© 1996 ACM 0-89791-794-4/96/0006...$3.50

respecting as much as possible the autonomy of sites. Four points characterise maximal autonomy: federation does not impose any modification on site software; communication between sites is minimized (particularly to communicate local control information to the GTM or to others sites); each site completely controls the commit or abort of its transactions (LTs or GSTs); and last, no restriction to data item access, belonging to whether local or global objects sub-sets, limits site transactions.

Two opposing approaches [2] have been proposed for concurrency control performed by the GTM. The first one consists of ensuring *global serializability* of all global and local transactions, thus preserving ACID properties for all transactions. Unfortunately, protocols providing this criterion are based on a more or less thorough knowledge of local control properties (loss of autonomy), increase the number of conflicts between GSTs and limit concurrency. The second approach consists of adopting more permissive correctness criteria, thus weakening ACID properties for global transactions. *Two-level serializability* (2LSR) [6] belongs to this approach. In this case, correctness of executions relies on a property of global transactions dependent on programmer strictness: the LDP property (Local Database Preserving). We propose here to get rid of this property while ensuring correctness of 2LSR executions, by taking into account the value dependencies.

The paper is organized as follows. Section 2 recalls two-level serializability criterion and the necessary hypotheses for correct executions. Section 3 describes the problem of taking value dependencies into account rather than imposing the LDP property. We propose defining precisely value dependencies and managing them in a flow graph. We prove that if the flow graph is acyclic then global executions are correct. Section 4 gives a practical application of this result. Section 5 compares our approach with other ones. Section 6 concludes this paper.

2. Two-level serializability and strong correctness

2.1 Principle

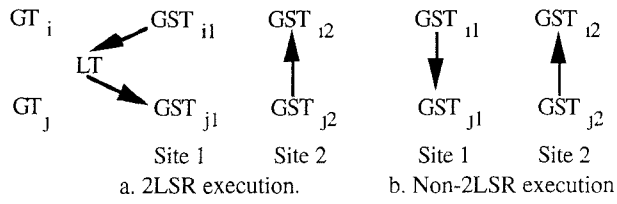
Two-level serializability exploits the two levels of control present in a multidatabase: local control (at each site) and global control (at the GTM).

Definition 1: [6]

A global schedule (execution) S is *two-level serializable* if:

- all the local schedules (S^{D_i}) are serializable (D_i is the database local to site i)
- the projection of S on the set of global transactions is serializable. ❑

Figure 2.a shows the archetype of a 2LSR execution, whereas Figure 2.b presents a non-2LSR execution. In a 2LSR execution, all indirect conflicts between global transactions, due to local transactions, are not taken into account. Indeed, in the projection of a global schedule on the set of global transactions, only direct conflicts appear.



a. 2LSR execution. b. Non-2LSR execution

Figure 2. 2LSR and non-2LSR executions.

Notation: ⟶ Dependency resulting from a conflict.

However, this criterion does not guarantee the same correction level as global serializability, since global transactions are not always executed in complete isolation. The notion of *strong correctness* has been introduced to characterize 2LSR schedules.

Definition 2: [2]

A global schedule is *strongly correct* if:

- it preserves database consistency (satisfies integrity constraints)
- it guarantees that all transaction views are consistent (all values read by transactions satisfy integrity constraints). ❑

Let us illustrate the difference between strong correctness and global serializability correctness by using an example.

Example. Consider a multidatabase consisting of two sites. Let D_1={a,c}, D_2={b}. Consider the following transaction programs: L: c:=c+a; a:=0

G_1: temp:=a+b

G_2: if a+b+c<2000 then b:=b+c; c:=0 endif

Consider the local schedules S_1 and S_2 resulting from the execution of L, G_1 and G_2 from the database state {a=500, b=500, c=500}:

S_1: $R_1(a,500)$ $R_L(a,500)$ $W_L(a,0)$ $R_L(c,500)$ $W_L(c,1000)$ $R_2(a,0)$ $R_2(c,1000)$ $W_2(c,0)$

S_2: $R_2(b,500)$ $W_2(b,1500)$ $R_1(b,1500)$

The final state is {a=0, b=1500, c=0}.

This execution is not globally serializable because at site 1 the order of serialization is GT_1 precedes GT_2 whereas it is GT_2 precedes GT_1 at site 2 (GT_1 and GT_2 are global transactions resulting from the execution of transaction programs G_1 and G_2). Whether this execution is strongly correct is subordinated to the existence and the nature of integrity constraints. Indeed, consider these two situations.

1 - There is a GIC between a and b: (a+b<2000). So, a and b are global objects and c is a local one. The execution is not then strongly correct, since GT_1's view does not satisfy the GIC. Indeed GT_1 sees temp=a+b=2000.

2 - There is no GIC between a and b, but there are two LICs: (a≥0) and (b≥0). All objects are local ones. The execution is in this case strongly correct since values read by GT_1 satisfy the LICs: a=500≥0 and b=1500≥0. The same goes for GT_2 that sees b with value 500. In this execution neither GT_1 nor GT_2 are isolated. Values of a and b seen by GT_1 (a=500, b=1500) would not be able to be obtained in a serial execution and correspond

with an intermediate database state. As a consequence, G_1's implicit constraint on a and b (to get their exact sum) is not satisfied [8], whereas it would be in a serializable execution. ❑

Unfortunately 2LSR schedules are not always strongly correct. Indeed, the 2LSR schedule of Figure 3 is not strongly correct as soon as the GIC (a+b<2000) exists.

2.2 Restrictive hypotheses

In order to have all 2LSR schedules strongly correct, three restrictive hypotheses have been proposed [6]. They concern the accesses to the objects, the nature of integrity constraints, and also the type of transactions.

Hypothesis H1: Transaction accesses conform to the $G_{rw}L_r$ model.

Hypothesis H2: No integrity constraint involves both local and global objects.

Hypothesis H3: Global transaction programs have no value dependency.

Hypothesis H1 means that the global read-write and local read ($G_{rw}L_r$) model is used. In this model, local transactions read and write local objects, but only read global objects (L_r); global transactions, in addition to reading and writing global objects, also read and write local objects (G_{rw}). This restriction imposed to LTs is justified by the incapacity of a transaction local to a given site to update objects located on a different site. Indeed, if it were possible for a local transaction to update a global object located on its site, a GIC could be violated.

In [6], it is proven that if the three hypotheses H1, H2 and H3 hold, then every 2LSR schedule is strongly correct.
The following example shows that when global transactions have value dependencies, H1 and H2 hypotheses are not sufficient for all 2LSR schedules to be strongly correct.

Example. Consider a multidatabase consisting of two sites. Let D_1={a,b,c}, D_2={d}. Suppose there are three LICs: (a>0->b>0), (c>0), (d>0), and let the initial state be {a=-1, b=-1, c=1, d=1}. Consider the following transaction programs: G_1: c:=d

G_2: if a>0 then d:=b endif

L: a:=1; if c>0 then b:=1 endif

Consider the local schedules S_1 and S_2 resulting from the execution of L, G_1 and G_2 from the initial state:

S_1: $W_L(a,1)$ $R_2(a,1)$ $R_2(b,-1)$ $W_1(c,-1)$ $R_L(c,-1)$

S_2: $W_2(d,-1)$ $R_1(d,-1)$

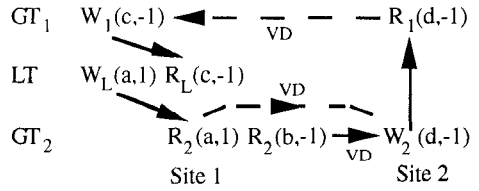


Figure 3.

Notation: —VD→ Value dependency.

In this example, all objects are local and there are no GICs. Transactions GT_1, GT_2 and LT resulting from the execution of transaction programs G_1, G_2 and L do naturally conform to the $G_{rw}L_r$ model. Nevertheless, execution is not strongly correct since the final state {a=1, b=-1, c=-1, d=-1} satisfies none of the local constraints. This is due to the existence of VDs. ❑

However, as noted in [2], having global transactions without VD (hypothesis H3) is more restrictive than necessary. A weaker hypothesis H4, replacing H3, has been proposed in [7]. It is based on the following definition. Let DS^{D_1} be the projection of the global database state DS on the local database D_1.

Definition 3:
A transaction program tp is *Local Database Preserving* if for all global database states DS_1 and DS_2 such that $\{DS_1\}tp\{DS_2\}$ we have: for all i, i=1,2,..,m, if $DS_1^{D_1}$ is consistent then $DS_2^{D_i}$ is consistent. ❑

In other words, an LDP transaction (program) preserves local consistency on each site regardless of the state of other sites. By definition, a local transaction (program) is LDP. As for a global transaction (program), it is shown in [2] that the absence of VDs ensures the LDP property. In the example of Figure 3, neither G_1 nor G_2 are LDP. Indeed, consider the database state DS_1={a=-1, b=-1, c=1, d=-1}. From this state, the execution of transaction program G_1 provides the state DS_2={a=-1, b=-1, c=-1, d=-1}. Although $DS_1^{D_1}$ is consistent, $DS_2^{D_1}$ is not. Consequently, G_1 is not LDP. The same goes for G_2, executed from the state DS_1={a=1, b=-1, c=-1, d=1}.

The weaker hypothesis H4 requires transactions to be LDP rather than without value dependency.

Hypothesis H4: Global transactions are Local Database Preserving (LDP).

The following theorem states conditions under which 2LSR schedules are strongly correct when using LDP property.

Theorem 1: [7]
Let S be a 2LSR schedule in the $G_{rw}L_r$ model (H1). If no integrity constraint is present between local and global objects (H2) and all transaction programs are LDP (H4), then S is strongly correct. ❑

Hypothesis H4 is not contrary to site autonomy, as local transaction (programs) are LDP. However, its main drawback is that it relies on rigorous programming. Indeed, determining LDP property for a global transaction program requires to consider all the possible database states (consistent or not) from which this program can be executed. Consequently, this cannot be done in a syntactic manner for the transaction program. The principal motivation of the paper is to suppress this drawback while conserving the strong correctness of 2LSR executions. Our basic approach is to accept value dependencies, and to

manage them in a graph which has to be acyclic. In a first step, we need to address the question of value dependencies.

3. Taking value dependencies into account

3.1 Value dependencies

The aim of our proposal is to remove hypothesis H4 that requires transactions to be LDP, while accepting the existence of VDs. In this way we propose to manage a graph of VDs, and to characterize a non strongly correct schedule by means of a cycle in this graph. Firstly, we precisely define value dependencies.

Definition 4: Value dependency.
Let $o_i(x)$ and $o_j(y)$ be two operations belonging to a global transaction program so that objects x and y are located at distinct sites. Objects x and y are connected by a *value dependency* if effects of operations correspond to either of the following situations:

a - the value given to x by $o_i(x)$ is a function of the value of y read by $o_j(y)$; this VD is expressed by a directed edge y-->x;

b - the execution of $o_i(x)$, and consequently the modification of x, depends on a predicate involving the value of y read by $o_j(y)$; this VD is expressed by a directed edge y-->x;

c - the objects x and y modified by o_i and o_j are involved in a global integrity constraint; this VD is expressed by an undirected edge y---x. ❑

The three types of VDs correspond in fact to three kinds of relations likely to exist between objects used by a transaction program:

a - x:=f(y);
b - if pred(y) then x:=f( ) endif;
c - x:=f( ), y:=g( ) and (x,y) ∈ GIC.

Dependencies of type a and b are directed, since the modification (possibly conditioned by a predicate) of an object depends on the value of another object. On the contrary, dependencies of type c are undirected since the modification of one object results in the other one being modified.

VDs are deduced from a syntactic analysis of the transaction program (contrary to LDP property). Consequently, it is possible that operations generating these VDs are not effectively executed (by the global transaction resulting from the execution of the transaction program). It may be the case if there are conditional statements. Nevertheless, the value dependency holds.

Example. Consider a multidatabase consisting of two sites. Let D_1={b,c,e}, D_2={a}. Let (e>0) be a local integrity constraint and ((a>0 or b>0)->c>0) a global one. Consequently e is a local object and a, b, c are global ones.

Consider the following transaction programs:
L: if b>0 then e:=c else e:=1 endif
G_1: b:=1; if a≤0 then c:=1 endif
G_2: a:=1;c:=1

Consider the local schedules S_1 and S_2 resulting from the execution of L, G_1 and G_2 from the database state {a=-1, b=-1, c=-1, e=1}:
S_1: $W_1(b,1)$ $R_L(b,1)$ $R_L(c,-1)$ $W_L(e,-1)$ $W_2(c,1)$
S_2: $W_2(a,1)$ $R_1(a,1)$

The value dependencies due to this execution are shown in Figure 4.

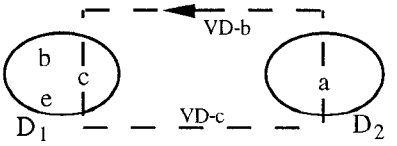


Figure 4. Value dependency graph.

Objects a and c are linked by a VD of type b (VD-b) because of the relation between the read operation on object a and the write operation on c (G_1). This illustrates the existence of a VD whereas the operation generating it, here $W_1(c,1)$, is not executed. As for the VD of type c (VD-c), it is due to both the existence of the GIC ((a>0 or b>0) -> c>0) and write operations on a and c (G_2).
This execution is 2LSR. However it is not strongly correct; the final database state is {a=1, b=1, c=1, e=-1}, which is inconsistent since the local integrity constraint is not satisfied. ❑

Let us note there is a cycle of value dependencies between the two sites (Fig. 4). We will show that the lack of such cycle guaranties strongly correctness of 2LSR schedules. So, value dependencies will be used in the next paragraph to build a graph called the flow graph.

3.2 The flow graph

We introduce the following notations:

$<_{vd}^{G_k}$: the set of VDs of type a and b induced by G_k,

$<>_{vd}^{G_k}$: the set of VDs of type c induced by G_k.

The VDs of a global transaction program are supposed known when the transaction begins its execution. This hypothesis will be discussed again in section 5.1.

Definition 5: Flow graph.
The flow graph, **FG** for short, is a graph <V, A_1, A_2> whose vertices are the databases local to the sites and the edges the VDs due to committed and active global transactions (programs). So:

$V=(D_1,D_2,...,D_m)$

$A_1=\{(D_i,D_j) \mid \exists G_k, x \in D_i$ and $y \in D_j$ so that $(x,y) \in <_{vd}^{G_k}\}$

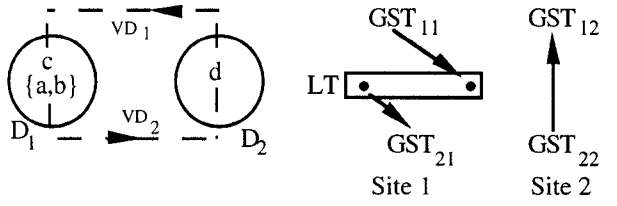
$A_2=\{(D_i,D_j) \mid \exists G_k, x \in D_i$ and $y \in D_j$ so that $(x,y) \in <>_{vd}^{G_k}\}$.
❑

The following theorem states conditions under which 2LSR schedules are strongly correct when using the flow graph.

Theorem 2:
Let S be a 2LSR schedule in the $G_{rw}L_r$ model. If no integrity constraint exists between local and global objects and FG is acyclic, then S is strongly correct. ❑

The proof (based on the one of the theorem 1) is given in Appendix. The interest of this theorem is that no restrictions need to be imposed on global transactions. The example below gives an illustration of the theorem.

Let us consider again the execution of Figure 3 schematized by Figure 5. This latter Figure shows dependencies due either to conflicts between transactions or to value dependencies. In this case the execution is 2LSR but the flow graph is not acyclic; therefore this execution is not considered as being strongly correct. Effectively, section 2 has shown that it was not strongly correct.



(a) Flow graph. (b) Conflict graph.

Figure 5.

The next section proposes a method that exploits this theorem.

4. Practical application

In order for conditions to be relevant to theorem 2, the flow graph, managed by the GTM, will be maintained acyclic by delaying the starting of global transactions introducing a cycle. For this, two sets are introduced:

Active: set of active (running) global transactions GT_k so that $<_{vd}^{Gk} U <>_{vd}^{Gk} \neq \emptyset$.

Waiting: set of global transactions waiting for execution.

Unfortunately, there are two problems of using theorem 2. The first one is that the flow graph can quickly become cyclic as it keeps on increasing. The method thus proposes reinitializing the flow graph FG every time the Active set becomes empty. This will be justified later. The second problem is that if $<_{vd}^{Gk} U <>_{vd}^{Gk}$ is cyclic then GT_k can never be executed because it always create a cycle in the flow graph. The method thus proposes executing GT_k alone. This will also be justified later.

The flow graph evolves in the three following circumstances: the starting, the commitment and the abort of a transaction. Note that global transactions without VDs are not controlled by the method.

Algorithm of flow graph management:

1 - *Initialization.*
FG:=<V, Ø, Ø>;
Active:=Ø;
Waiting:=Ø.

2 - *Starting of a global transaction GT_k having VDs.*
if FG U($<_{vd}^{Gk} U <>_{vd}^{Gk}$) is acyclic or Active = Ø then
A_1:=$A_1 U <_{vd}^{Gk}$;
A_2:=$A_2 U <>_{vd}^{Gk}$;
Active:=ActiveU{GT_k};
Execution of GT_k is started;
else GT_k is blocked; Waiting:=WaitingU{GT_k} endif.

3 - *Commitment of a global transaction GT_k having VDs.*
Active:=Active - {GT_k};
if Active = Ø then
FG:=<V, Ø, Ø>;
For every $GT_l \in$ Waiting the starting phase is performed again.
endif.

4 - *Abort of a global transaction GT_k having VDs.*
Active:=Active - {GT_k};
A_1:=A_1 - ($<_{vd}^{Gk}$);
A_2:=A_2 - ($<>_{vd}^{Gk}$);
if Active = Ø then FG:=<V, Ø, Ø> endif;
For every $GT_l \in$ Waiting the starting phase is performed again. ❑

For this method to be correct, it is necessary to add an hypothesis for local executions: strong serializability [2].

Definition 6:
A local execution is said to be *strongly serializable* if:
when the end (commitment or abort) of T_1 precedes the starting of T_2 then T_1 precedes T_2 in the local serialization order. ❑

Most of local concurrency controls (2PL, timestamps ordering and optimistic control) guarantee this property. Only controls based on the Thomas' rule or on graph testing do not provide this property.
Note that this property of local control does not enable the serialization order of transactions being executed in parallel to be deduced. However, it is this property that allows VDs to be forgotten at the end of the execution of a set of active and concurrent transactions, and GT with cyclic VDs to be executed alone.

Execution of Figure 5 illustrates this. Suppose that GT_2 has been executed before the beginning of GT_1 and that its commitment has caused the reinitialization of FG (because Active was empty). This results in VD_2 being eliminated. In this situation, we have already observed (cf. 3.2) that the execution of GT_1 can lead to a schedule that is not strongly

correct. The indirect conflict via LT between GT_1 and GT_2 creates GT_1->LT and LT->GT_2 dependencies (LT at site 1 began its execution before GST_{21} and finished it after the beginning of GST_{11}), and is at the origin of the incorrectness. If local executions are strongly serializable, this conflict cannot arise for global transactions not running concurrently. Therefore, if there is an indirect conflict between GT_2 and GT_1 it will necessarily follow the dependency GT_2->GT_1. Thus, execution will be globally serializable and consequently strongly correct despite the cycle of value dependencies. More generally, thanks to strong serializability property, global transactions executed serially are globally serializable. Execution is thus strongly correct even if VDs generate a cycle. This is the reason why VDs can be omitted. With this method, a GT with cyclic VDs is executed alone. Consequently, it is serialized with all others GTs and the cycle of VDs it introduces does not create inconsistency.

Finally, note that the commit or abort operations of a GT do not result in the same actions on the flow graph. In fact, when a GT is aborted its VDs can be immediately removed from FG because an aborted transaction has no effect on database objects. It results that, whenever a GT is aborted, the starting phase is performed again for all GT waiting for execution. On the contrary, when a GT is committed its VDs cannot be immediately removed from FG if Active is not empty as we saw before.

5. Comparison with other works

5.1 Discussion about hypotheses

We have seen that flow graph management allows us to obtain 2LSR schedules being strongly correct, in presence of value dependencies. This result is based on the hypothesis that the local schedules are strongly serializable. At first glance it might appear that this supposition is too restrictive in terms of site autonomy. This is not the case, however, as shown below.

Let us consider protocols [7] that provide 2LSR schedules. The first one (called 2LPL) supposes that all sites use a two phase locking protocol (2PL). The use of 2PL ensures transactions are serialized when they reach their maximum locked point [1]. Since this point is inevitably reached before commit, any local schedule generated by the 2PL protocol is still strongly serializable. The second one (called G2LPL) forces global transactions to write a global object, called ticket [5], when sites do not use the 2PL protocol. This results in all global transactions directly conflicting on this object, thus fixing their serialization order. In particular, two global transactions serially executed on a site will be serialized in that order (thus ensuring that global transactions are strongly serializable). The hypothesis relative to strong serializability of local control, required by the flow graph management in our method, is thus not restrictive in comparison with these protocols.

Let us now consider taking into account value dependencies. Our method supposes VDs to be known at the beginning of global transactions. Not only our method needs this hypothesis. *A priori* knowledge of VDs is also needed by methods based on the LDP property. If the LDP property of global transactions has to be controlled by the GTM, this control cannot happen during transaction processing. For instance in Figure 4, the execution of the (non-LDP) global transaction G_1 will result in an inconsistent writing by the local transaction LT. Since the GTM ignores local transactions, it cannot then remove the inconsistency. As a result, the LDP control has to be done before the beginning of global transactions. We know that the lack of VD in a global transaction, ensures the LDP property. Unfortunately, we do not know a simple method which, even in presence of VDs, ensures LDP property for a global transaction. The alternative is then as follows: either the LDP property is *a priori* controlled by the GTM, in a restrictive way, when verifying lack of VD; or this property is ensured by the programmer, compromising in case of error the database consistency. In both cases, the *a priori* knowledge of VDs for a transaction is needed.

To sum up, the hypotheses needed by our flow graph management (*a priori* knowledge of VDs, strong serializability) are also implicitly supposed by methods based on LDP property.

At last, let us consider protocol [4] which provides QuasiSerializable schedules (a particular case of 2LSR schedules). This protocol manages a graph (similar to ours) called the Information Flow Graph (IFG), which is also based on the *a priori* knowledge of VDs. However, the IFG relies on a VD definition which is not so complete as ours, since only VDs of type a and b are considered. It results that the IFG can only be used when there is no GIC. Compared with this protocol, our approach allows the existence of GICs between objects by extending the definition of VDs.

5.2 LDP and non-LDP transactions

We next show how our method controls global transaction programs according to their type (LDP or non- LDP). In this way, we investigate schedules generated by flow graph management. FG is managed so that the Active set is periodically reset. In the meantime, this set contains the active global transactions that can be executed concurrently. Figure 6 shows the evolution of the Active set during the execution. A change in numbering expresses that Active becomes empty.

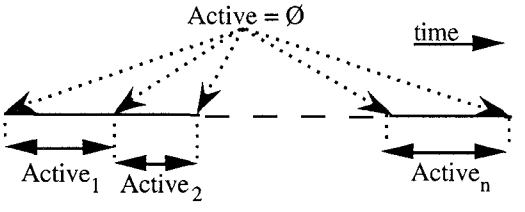


Figure 6. Evolution of Active set.

This presentation points out the following properties.
$\forall GT_i \in Active_k$ and $\forall GT_j \in Active_q$:

- GT_i is globally serialized before GT_j if k<q
- GT_i is globally serialized after GT_j if k>q
- GT_i and GT_j are not necessarily globally serializable, but the schedule is 2LSR if q=k.

In other words, flow graph management generates 2LSR schedules for global transactions executed concurrently, and globally serializable schedules for global transactions executed serially.

Let us consider again the example of Figure 3 (section 2.2) to illustrate how non-LDP transactions (programs) are managed. Both non-LDP transactions GT_1 and GT_2 will be accepted, but one of them will be delayed until the termination of the other. Therefore, transaction execution will not be able to happen as in the Figure 3 (because of a cycle in FG). Both transactions will in fact be globally serialized. More generally, in our method, a non-LDP transaction which would introduce a cycle of VDs in FG, will be globally serialized after concurrent transactions in Active set.

Concerning an LDP global transaction program, two cases have to be considered.

1 - If the global transaction program has no value dependency, it can be executed concurrently, without being delayed;

2 - If it has some value dependencies, although being LDP, it risks being delayed.

The following example illustrates this second point.

Example. Consider a multidatabase located at two sites. Let $D_1=\{a,b\}$ and $D_2=\{c\}$. There is no integrity constraint on these objects, so all objects are local ones. The transaction programs are the following: L: temp:=a+b

G_1: c:=a; a:=0

G_2: b:=c

Note that the global transaction programs, although being LDP (since there is no integrity constraint) both introduce some value dependencies. For G_1, the VD-a concerns a and c, for G_2, c and b. Consider the local schedules S_1 and S_2 resulting from the execution of L, G_1 and G_2 from the initial state {a=1, b=0, c=0}:

S_1: $R_L(a,1)$ $R_1(a,1)$ $W_1(a,0)$ $W_2(b,1)$ $R_L(b,1)$

S_2: $W_1(c,1)$ $R_2(c,1)$

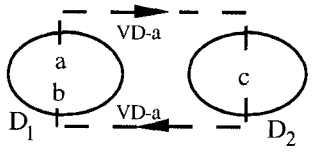


Figure 7. Execution of LDP transaction programs.

This schedule is 2LSR. As it concerns the $G_{rw}L_r$ model and LDP transaction programs, this schedule is strongly correct. With our method, GT_1 and GT_2 (resulting from the executions of G_1 and G_2) could not be executed concurrently because of a cycle in FG. None of the transactions would be rejected, but one of them would be delayed until the commit of the other, leading to a global serializable schedule. In this case, concurrency among global transactions is decreased. ❑

To sum up, flow graph management allows more global transactions to be executed since non-LDP transactions can be also taken into account, at the price of reducing concurrency among them in certain cases.

5.3 Difficulties in capturing LDP property

Example. Consider a multidatabase located at two sites. Let $D_1=\{a,b,e\}$ and $D_2=\{c\}$. The global integrity constraints are (c>0->a>0), (e>0->c>0), (e>0->a>0); the local constraint is (b>0). Consequently, a, c, e are global objects while b is a local one. The initial state is {a=-1, c=-1, e=-1, b=1}. The transaction programs are the following:

G_1: a:=1; c:=1

G_2: if c>0 then e:=1 else e:=-1 endif

L: temp:=a; if e≤0 then temp:=1 endif; b:=temp

Concerning G_2, it would appear that it is LDP, as it seems to maintain the database consistency, whatever the consistent value of c. However, consider the following local schedules resulting from transaction programs execution from the initial state:

S_1: $R_L(a,-1)$ $W_1(a,1)$ $W_2(e,1)$ $R_L(e,1)$ $W_L(b,-1)$

S_2: $W_1(c,1)$ $R_2(c,1)$

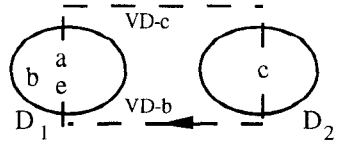


Figure 8. Execution of transactions with VDs.

This schedule, although being 2LSR, is not strongly correct, as the final state {a=1, b=-1, e=1, c=1} does not satisfy the constraint (b>0). This is due to the fact that the transaction program G_2 is not LDP. Indeed, GT_2 is executed from the database state $DS_1=\{a=-1, c=1, e=-1, b=1\}$ which is consistent with regard to site 1. Consequently, if G_2 was LDP, GT_2 would leave the database in a consistent state with regard to site 1. This is not the case, since local transaction LT, which follows it in the local serialization order, reads from the state $DS_2=\{a=-1, c=1, e=1, b=1\}$ an inconsistent value of e. ❑

This example shows the difficulty of characterizing LDP property for a global transaction program. The ambiguity of such situations is automatically resolved by our method, while accepting all the transactions and managing their value dependencies by using the flow graph.

6. Conclusion

This paper addresses the theoretical problem of maintaining database consistency in multidatabase systems, when global transaction programs exhibit some value dependencies. In

this context, we propose to extend the notion of value dependency, by taking into account the global integrity constraints defined over the database. A method allowing value dependencies and ensuring 2LSR schedules to be strongly correct, has been proposed and proved. It is based on the use of a flow graph. We have shown that the hypotheses needed by flow graph management are also implicitly used by methods based on the LDP property. The flow graph management relaxes the constraints on global transactions that are no longer restricted to being LDP.

At last, it is easier to get and to manage VDs of a transaction program than to prove it is LDP. Indeed, extracting VDs from a transaction program can be done in a syntactic manner whereas determining the LDP property is not.

Bibliography

[1] Bernstein P. A., Hadzilacos V., Goodman N.; *Concurrency Control and Recovery in Database Systems*; Addison Wesley Publishing, Reading, MA, 1987.

[2] Breitbart Y., Garcia-Molina H., Silberschatz A.; *Overview of Multidatabase Transaction Management*; VLDB Journal, vol. 1, no. 2, October 1992.

[3] Du W., Elmagarmid A.K ; *Quasi Serializability: a Correctness Criterion for Global Concurrency Control in InterBase*; Proc. 5th International Conference on Very Large Data Bases, Amsterdam, 1989, pp. 347-355.

[4] Elmagarmid A.K., Du W.; *Maintaining HDDBS Consistency: the Quasi Serializability Approach*; Univ. of Purdue, Department of Computer Sciences, Technical Report, CSD-TR-1017, 1990.

[5] Georgakopoulos D., Rusinkiewicz M., Sheth A.; *On Serializability of Multidatabase Transactions Through Forced Local Conflicts*; Proc. 7th International IEEE Conference on Data Engineering, Kobe, Japan, April 1991, pp. 314-323.

[6] Mehrotra S., Rastogi R., Korth H., Silberschatz A.; *Non-Serializable Executions in Heterogeneous Distributed Database Systems*; Proc. 1st International Conference on Parallel and Distributed Systems (PDIS), Miami Beach, December 1991, pp. 245-252.

[7] Mehrotra S., Rastogi R., Korth H., Silberschatz A.; *Maintaining Database Consistency in Heterogeneous Distributed Database Systems*; Univ. of Texas, Austin, Department of Computer Sciences, Technical Report, TR-91-04, 1991.

[8] Mehrotra S , Rastogi R., Korth H., Silberschatz A.; *Relaxing Serializability in Multidatabase Systems*, 2nd International Workshop on Research Issues on Data Engineering: Transaction and Query Processing, Mission Palms, Arizona 1992.

Appendix : Proof of theorem 2

Definitions and lemmas in italic are described in [7]. They are used to prove theorem 2 and to this end, they are just mentioned and not proven. They are based on the following notations.

τ : set of transactions of a global schedule S.

τ_L : subset of local transactions, all sites included.

τ_G : subset of global transactions.

$\tau_W(d,S)$: subset of transactions of S writing objects of d.

Read(t): set of object values read by t.

Read(t^{D_i}): set of D_i's object values read by t.

WS(t): set of objects written by t.

WS(t^{D_i}): set of D_i's objects written by t.

Write(t): set of object values written by t.

Write(t^{D_i}): set of D_i's object values written by t.

Legal(DS_1,S):

A database state DS_1 is legal with respect to an operation o_i, if it is possible to execute o_i from DS_1. Thus **legal(DS_1,o_i)** if: either action(o_i)=w or if action(o_i)=r, then (object(o_i),value(o_i)) $\in DS_1$.

A database state DS_1 is legal with respect to a schedule $S=(o_1,o_2,\ldots,o_p)$ if it is possible to execute $o_1,o_2,\ldots,o_p$ from DS_1. Thus **legal(DS_1,S)** if: legal(DS_1,o_1) and if p>1, then legal(DS'_1,o_2,...,o_p) where $\{DS_1\}o_1\{DS'_1\}$. ❑

State(t_j,D_i,S,DS_1):

Let S be a schedule, t be a transaction in S, $D_1 \subset D$ and DS_1 be a database state such that legal(DS_1,S). The state seen by t with respect to D_i exists if $(S^{\tau'})^{D_1}$ is serializable where $\tau_W(D_i,S) \subseteq \tau'$. Without loss of generality let $t_1,t_2,\ldots,t_n$ be a serialization order of transactions in $(S^{\tau'})^{D_i}$. The state of an arbitrary transaction t_j, j=1,2,...,n, is defined recursively as follows:

State(t_j,D_i,S,DS_1)= either $DS_1^{D_i}$ if j=1

or State(t_{j-1},D_i - WS(t_{j-1}),S,DS_1) U write($t_{j-1}^{D_i}$) if j>1. ❑

State(t_j,D_i,S,DS_1) is the set of D_i's object values may be read by t_j, during the execution of the schedule S, starting from the initial database state DS_1. State(t_j,D_i,S,DS_1) depends on the values written by $U_k t_k$, k=1...j-1, on D_i's data items. t_j may not read all the values of state(t_j,D_i,S,DS_1), but values it will read on D_i's data items will be included in state(t_j,D_i,S,DS_1).

In [7], the proof of theorem 1 relies on a lemma that has to be proven again using our new hypothesis, in order to prove theorem 2.

Lemma 1:

Let S be a schedule and DS_1 be a database state such that *legal(DS_1,S)* and $\{DS_1\}S\{DS_2\}$. If for all i, i=1..m:

- S^{D_i} is serializable (let $t_1..t_n$ be a serialization order of transactions in S^{D_i} - of any local and global transactions)
- FG is acyclic
- $DS_1^{D_1}$ is consistent

then $DS_2^{D_1}$ is consistent and state(t_j,D_i,S,DS_1) is consistent for all j=1,...,n. ❑

The proof of lemma 1 relies on lemma A specially defined for our new hypothesis. In order to simplify its proof, we make an extrapolation of VDs. Suppose G is a global transaction program with a VD between objects x and y located at site 1 and 2. If operations of G, at the beginning of this VD, are executed, they belong to two different GSTs, say GST_1 and GST_2. So, by extrapolation, we say that there is a VD between GST_1 and GST_2 even if operations of G creating the VD are not effectively executed.

Lemma A:

Let S be a schedule and DS_1 be a consistent database state such that legal(DS_1,S). Let G_0 be a global transaction such that $t_k=GST_{0i}$ and $t_l=GST_{0j}$. Let o_i and o_j be two operations such that $o_i(x) \in t_k$, $o_j(y) \in t_l$ and $(y,x) \in <^{G0}_{vd} U <>^{G0}_{vd}$. If:

- state(t_k,D_i,S,DS_1) is consistent
- state(t_l,D_j,S,DS_1) is consistent
- FG is acyclic

then state(t_k,D_i,S,DS_1) U state(t_l,D_j,S,DS_1) is consistent. ❑

If the flow graph is not managed, this result is not immediate as can be seen by the execution in Figure 4. In this case there are two integrity constraints, ((a>0 or b>0) -> c>0) being a global one and (e>0) being a local one. The initial database state is {a=-1, b=-1, c=-1, e=1} In this situation state(GST_{12},D_2,S,DS_1)={a=1} is consistent because from the view point of site 2 there is no integrity constraint on a. In the same way, state(GST_{11},D_1,S,DS_1) ={b=-1, c=-1} is consistent as with respect to site 1 the global integrity constraint is reduced to (b>0 -> c>0). Yet, state(GST_{12},D_2,S,DS_1) U state(GST_{11},D_1,S,DS_1)={a=1, b=-1, c=-1} is not consistent as the global integrity constraint is violated.

Lemma A states under what conditions state(GST_{12},D_2,S,DS_1)U state(GST_{11},D_1,S,DS_1) may be consistent.

Proof of lemma A:

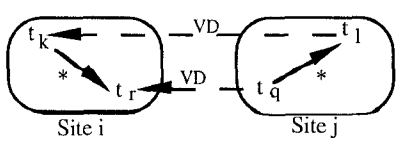
Hypotheses: t_k=GST_{0i}, t_l=GST_{0j}, $\exists$ $o_i(x) \in t_k$, $\exists$ $o_j(y) \in t_l$ such that $(y,x) \in <_{vd}^{G0} \cup >_{vd}^{G0}$ DS_1, state(t_k,D_i,S,DS_1) and state(t_l,D_j,S,DS_1) are consistent.

If state(t_k, D_i,S,DS_1)Ustate(t_l,D_j,S,DS_1) is not consistent, inconsistency is due to the violation of a global integrity constraint gic_1. Consequently, there is a global transaction GT_1 so that it updates data items of this constraint, leading to inconsistency for GT_0. Let t_r be GST_{1i} and t_q be GST_{1j}. We will demonstrate by a reduction to absurdity, that state(t_k,D_i,S,DS_1)Ustate(t_l,D_j,S,DS_1) is always consistent. The proof is based on the nature of the value dependency between t_r and t_q.

a - There is no VD between t_r and t_q: consequently, GT_1 cannot access gic_1's data items at the two sites, unless it only performs reads (definition of value dependency). If GT_1 only reads data item of gic_1, it cannot induce an inconsistency for GT_0. If GT_1 writes some objects belonging to the constraint, it performs updates at just one site, without having knowledge of gic_1's data items values at the other site. Therefore, values written by GT_1, at one of the sites, must always respect the constraint whatever the values of other data items of the constraint at the other site. Then, G_1 cannot induce an inconsistency for GT_0. Thus, state(t_k,D_i,S,DS_1) Ustate(t_l,D_j,S,DS_1) is consistent

b - If $(y,x) \in <>_{vd}^{G0}$ (the value dependency of G_0 is a undirected VD): consequently, whatever the value dependency between t_r et t_q, there will always be a cycle in FG if GT_0 and GT_1 are executed together However, this situation is proscribed, thus GT_0 and GT_1 are not executed at the same time. So, state(t_k,D_i,S,DS_1) Ustate(t_l,D_j,S,DS_1) cannot be inconsistent because GT_0 is the only one accessing gic_1's data items at site i and site j at a time. Therefore, if the value dependency of G_0 is a undirected VD, then state(t_k,D_i,S,DS_1) U state(t_l,D_j,S,DS_1) is always consistent.

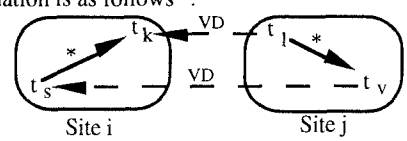
c - If $(y,x) \in <_{vd}^{G0}$ (the value dependency of G_0 is directed from y to x): consequently, in FG there is an arrow from D_j to D_i. So, the only dependency between t_r and t_q is a VD directed in the same sense as the one of G_0. This situation is pictured below. Arrows with an asterisk reflect local serialization orders between GT_0 and GT_1[1]



Transaction GT_1 cannot write gic_1's data items both at site i and site j (because in this case the value dependency would be undirected). Because of the direction of the VD, only gic_1's data items of site i are written by GT_1 Consequently, GT_1 makes these updates knowing values of others objects of the constraint at site j (as the VD is directed from D_j to D_i). But it does not modify data items of the constraint at site j. Therefore, object values of gic_1 taken into account by GT_0 at site j are initial values (those from DS_1). For state(t_k,D_i,S,DS_1) U state(t_l,D_j,S,DS_1) to be consistent, values of gic_1's data items at site i, taken into account by t_k, have to be consistent with initial values at site j. We next need to fix which values of gic_1 correspond to the state of execution of t_k.

- Either they are initial values (from DS_1), thus state(t_k,D_i,S,DS_1) U state(t_l,D_j,S,DS_1) is naturally consistent.
- Or there is a global transaction GT_2 so that t_s=GST_{2i}, t_v=GST_{2j} and such that state(t_k,D_i,S,DS_1) results from the execution of t_s Two cases are considered according to whether G_2 has a VD between t_s and t_v or not.

- If G_2 has no value dependency, then updates of t_s on gic_1 are necessarily consistent with initial values of gic_1 at site j. This is because the global transaction GT_2 is consistent and does not update gic_1's data items at site j (otherwise, there would be a undirected VD between t_s and t_v).
- If there is a value dependency between t_s and t_v, it can only be in the sense t_v->t_s. Effectively, all other dependency would create a cycle in the flow graph due to the dependency between t_l and t_k in G_0. The situation is as follows[2]:



Thus, t_s writes gic_1's data items at site i, whereas t_v cannot write them at site j. Now, t_l does not update gic_1's objects at site j. Thus, values written at site i by t_s are necessarily compatible with initial object values of the constraint at site j (consistency of the global transaction). Therefore, state(t_k,D_i,S,DS_1) U state(t_l,D_j,S,DS_1) is consistent.

Thus, whatever the situation state(t_k,D_i,S,DS_1)Ustate(t_l,D_j,S,DS_1) will always be consistent if there is a value dependency between t_k and t_l. ❑

Proof of lemma 1:

We begin by showing that state(t_j,D_i,S,DS_1) is consistent for all j=1,2,..,n. The proof is by induction on j.

Base: j=1 state(t_j,D_i,S,DS_1)= $DS_1^{D_i}$ consistent.

Induction: assume state(t_k,D_i,S,DS_1) is consistent. Prove the property at level k+1. Let $DS_3^{D_i}$=state(t_k,D_i,S,DS_1) and legal(DS_3,t_k). So, $DS_3^{D_i}$ is consistent because state(t_k,D_i,S,DS_1) is consistent. Let $\{DS_3\}t_k\{DS_4\}$. As $DS_4^{D_i}$ =state(t_{k+1},D_i,S,DS_1), we must prove that $DS_4^{D_i}$ is consistent.

(1) If t_k is the result of the execution of an LDP global transaction program then by definition $DS_4^{D_i}$ is consistent.

(2) If t_k is not the result of the execution of an LDP global transaction program, then there exists t_l such that t_k=GST_{0i}, t_l=GST_{0j}, and $\exists$ $o_i(x) \in t_k$, $\exists$ $o_j(y) \in t_l$ such that $(y,x) \in <_{vd}^{G0} \cup >_{vd}^{G0}$. Thus, because of the VD, for $DS_4^{D_i}$ to be

[1] Note that in the case of 2LSR schedule, at least one of the two orders is due to an indirect conflict via a local transaction.

[2] Necessarily $t_l \overset{*}{\rightarrow} t_v$, because if not $t_v = t_q$ thus $t_s = t_r$ and consequently t_s is not serialized before t_k but after

consistent, state(t_k,D_i,S,DS_1) U state(t_l, D_j,S,DS_1) and state(t_l,D_j,S,DS_1) have to be consistent in addition to state(t_k,D_i,S,DS_1) consistent. First, fix state(t_l,D_j,S,DS_1) to be consistent.

• Either t_l is the first transaction in the serialisation order of S^{D_j}, thus state(t_l,D_j,S,DS_1)= $DS_1^{D_j}$ is consistent.

• If not, induction is repeated for t_l: assume state(t_n,D_j,S,DS_1) is consistent for n=1..l-1. So, state(t_{l-1},D_j,S,DS_1) is consistent. Then there are two cases. Either t_{l-1} is the result of the execution of an LDP global transaction program, thus state(t_l,D_j,S,DS_1) is consistent and by lemma A, state(t_k,D_i,S,DS_1)Ustate(t_l,D_j,S,DS_1) is consistent. Or there exists t_h such that t_h=GST_{lz} and t_{l-1}=GST_{lj}. For state(t_l,D_j,S,DS_1) to be consistent, state(t_h,D_z,S,DS_1) and state(t_h,D_z,S,DS_1) U state(t_{l-1},D_j,S,DS_1) have to be consistent in addition to state(t_{l-1},D_j,S,DS_1). If t_h is the first transaction in the serialization order of S^{D_z}, then state(t_h,D_z,S,DS_1) is consistent and by lemma A state(t_h,D_z,S,DS_1) U state(t_{l-1},D_j,S,DS_1) is consistent. Otherwise, induction is repeated for t_h.

In fact the induction is repeated until a transaction is the result of an LDP global transaction program execution or first serialized in the local serialization order. This transaction always exists because of the acyclicity of the flow graph FG. Effectively, the number of sites is finite; so, if there was no transaction, resulting of an LDP global transaction program execution, during the iteration (or no transaction first serialized in the local serialization order), a value dependency would create a cycle in FG. This then is the contradiction in the hypothesis.

When the transaction resulting of the execution of an LDP global transaction program (or first serialized) has been found, lemma A is used to return the iteration of induction. So, we prove that state(t_l,D_j,S,DS_1) is consistent and by lemma A that state(t_k,D_i,S,DS_1)Ustate(t_l,D_j,S,DS_1) is consistent (because there is a value dependency in G_0). Therefore, t_k will produce a consistent state, i.e. $DS_4^{D_i}$ will be consistent.

State(t_m,D_i,S,DS_1) can be shown consistent. Thus, by the same argument as before, $DS_2^{D_i}$ can be proved consistent. Effectively, state $DS_2^{D_i}$ is produced by the later transaction serialized in S^{D_i} which is t_m. ❑

Now we can prove the theorem 2.

Proof of theorem 2:

Let DS_1 be a consistent database state such that legal(DS_1,S). Let {DS_1}S{DS_2}. In order to prove that S is strongly correct we need to show that DS_2 is consistent and that transaction views are consistent.

Since no integrity constraints are present between local and global objects, the integrity constraints can be viewed as IC=$L_1 \wedge \ldots \wedge L_m \wedge G$ where L_i is the set of local integrity constraints on LD_i (local objects at site i; $LD_i \subseteq D_i$) and G is the set of global integrity constraints on GD (global objects at all sites; GD $\subset$ D). The demonstration is in two steps; the first concerns the consistency of local objects and views of local transactions, as well as the consistency of views of local objects by global transactions (based on lemma 1) and the second concerns the consistency of global objects and views of global objects by global transactions.

(1) We use the fact that the flow graph is acyclic to prove that $L_1,\ldots,L_m$ are preserved by S, that local transactions read consistent local and global data items and that global transactions read consistent local data items. Since DS_1 is consistent, by *lemma 3*, $DS_1^{D_i}$ is consistent. Since S^{D_i} is serializable (let $t_1..t_n$ be a serialization order of transactions in S^{D_i}), FG acyclic, by lemma 1, $DS_2^{D_i}$ is consistent and state(t_j,D_i,S,DS_1) too for all j=1,..,n. Since $LD_i \subseteq D_i$ and $DS_2^{D_i}$ is consistent, $DS_2^{LD_i}$ is consistent. Since read(t^{D_i}) $\subseteq$ state(t,D_i,S,DS_1), read(t^{D_i}) is consistent for all $t \in \tau$. Since $LD_i \subseteq D_i$, read(t^{LD_i}) for all $t \in \tau_G$ is consistent. Also, since local transactions access objects at a single site, read(t) for all $t \in \tau_L$ is consistent because read(t) = read(t^{D_i}) for all $t \in \tau_L$ being executed at site i.

(2) We now show that G is preserved by S.

Since only global transactions write global objects, τ_W(GD,S) $\subseteq \tau_G$. Since S^{τ_G} is serializable $(S^{\tau_G})^{GD}$ is serializable. As shown above read(t^{LD_i}) is consistent, $t \in \tau_G$. Since $LD_i \cap LD_j = \emptyset$, i≠j, by *lemma 2*, $\cup_{i=1}^{m}$ read(t^{LD_i}) = read($t^{\cup_{i=1}^{m} LD_i}$) is consistent. Since DS_1 is consistent, by *lemma 3*, DS_1^{GD} is consistent. Thus by *corollary 1*, DS_2^{GD} is consistent and state(t,GD,S,DS_1) for all t, $t \in \tau_G$, is consistent. Since read(t^{GD}) $\subseteq$ state(t,GD,S,DS_1) and read($t^{\cup_{i=1}^{m} LD_i}$), $t \in \tau_G$, are consistent, by *lemma 2*, read(t) for all $t \in \tau_G$ is consistent.

Thus DS_2^{GD} and $DS_2^{LD_i}$ for all i=1,2,...,m are consistent. Hence by *lemma 3*, DS_2 is consistent.

Moreover, for all $t \in \tau_G$ read(t) is consistent, and for all $t \in \tau_L$ read(t) is consistent. Thus S is strongly correct. ❑

Lemma 2:

Let IC=$C_1 \wedge \ldots \wedge C_m$, where $C_1,\ldots,C_m$ are defined over data items in $d_1,d_2,\ldots,d_m$, where $d_i \cap d_j = \emptyset$ for all i≠j. Let $d'_i \subseteq d_i$ and DS be a database state. $DS^{d'_i}$, for all i, i=1,2.. .,m is consistent if and only if $\cup_{i=1}^{m} DS^{d'_i}$ is consistent. ❑

Lemma 3:

Let IC=$C_1 \wedge \ldots \wedge C_m$, where $C_1,\ldots,C_m$ are defined over data items in $d_1,d_2,\ldots,d_m$. Let DS be a database state DS^{D_i}, for all i, i=1,2,...,m is consistent if and only if DS is consistent. ❑

Corollary 1:

Let IC=$C_1 \wedge \ldots \wedge C_m$, where $C_1,\ldots,C_m$ are defined over data items in $d_1,d_2,\ldots,d_m$,d such that $d_i \cap d = \emptyset$, i=1,2,...,m. Let S be a schedule and DS_1 be a database state such that legal(DS_1,S) and {DS_1}S{DS_2}. Let τ' be any set of transactions such that τ_W(d,S) $\subseteq \tau'$. If: - $(S^{\tau'})^d$ is serializable (let $t_1,t_2,\ldots,t_n$ be a serialization order in $(S^{\tau'})^d$), - read($t^{d'}$) is consistent, for all t, $t \in \tau_W$(d,S) and d'= $\cup_{i=1}^{m} d_i$, and - DS_1^d is consistent,
then DS_2^d is consistent and state(t_i,d,S,DS_1) is consistent for all i, i=1,2,...n. ❑