1990

Maintaining Transaction Consistency in HDDBSs Using Quasi Serializable Executions

Weimin Du

Ahmed K. Elmagarmid
Purdue University, ake@cs.purdue.edu

Report Number:
90-969

MAINTAINING TRANSACTION CONSISTENCY IN HDDBSs USING QUASI SERIALIZABLE EXECUTIONS (EXTENDED ABSTRACT)

Weimin Du
Ahmed Elmagarmid

CSD-TR-969
March 1990

Maintaining Transaction Consistency in HDDBSs Using Quasi Serializable Executions*
(Extended Abstract)

Weimin Du and Ahmed Elmagarmid
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
{du,ake}@cs.purdue.edu

1 Introduction

Database Consistency is compromised by improper execution/interleaving of transactions. Transaction consistency addresses one aspect of database consistency which is caused by incorrect scheduling of transactions. Schedules produced by a given concurrency control algorithm are normally checked for correctness using serializability. An execution is serializable if it is equivalent to a serial execution of the same set of transactions.

Despite its elegance and success in traditional (homogeneous) distributed database systems, it is not, however, adequate for heterogeneous distributed database systems (HDDBSs), due to both heterogeneity and autonomy of local database systems (LDBSs) [DELO89] [DEK90].

In [DE89], we proposed quasi serializability as a correctness criterion for concurrency control in HDDBSs. A global execution of a set of local and global transactions is quasi serializable if local executions are all serializable and it is equivalent to a quasi serial execution in which the global transactions are executed sequentially. The quasi serializability approach is different from that of serializability in that it does not have any requirements on executions of local transactions at different sites. Therefore, quasi serializability of executions can be effectively maintained without violating local autonomy [ED90] [DE90]. On the other hand, local transactions at different sites

*This work is supported by a PYI Award from NSF under grant IRI-8857952 and grants from AT&T Foundation, Tektronix, SERC and Mobil Oil.

may interfere with each other in an undesirable fashion (e.g., mutually).

In an HDDBS, local transactions represent applications of users at different organizations and therefore are originally independent. No coordination between them is necessary if they operate on different databases. Even after integration, executions of local transactions at different sites may still be independent. Such independence is obviously helpful in maintaining transaction consistency. Unfortunately, global transactions may introduce interference between executions of local transactions at different sites. It is possible to prevent the interference by scheduling transactions properly (e.g., serializably). Such scheduling is, however, very difficult (if not impossible) in HDDBSs. On the other hand, the undesirable interference can be explicitly prevented at the global level. In other words, it is possible to submit global transactions in such a way that they do not introduce any undesirable interference between local transactions at various sites.

The basic idea of the quasi serializability approach is to take advantage of the hierarchical structure of HDDBSs and the independency of local executions to simplify the transaction consistency problem. Local database management systems guarantee the serializability of local executions, while the global database management system controls the submission and execution of global transactions only. Possible interference between local executions are controlled explicitly.

In this paper, we study the problem of maintaining transaction consistency using quasi serializable executions. The main results of the paper are: (1) aspects of transaction consistency that can be effectively maintained by quasi serializable executions and (2) restrictions and techniques to prevent possible violation of those aspects of transaction consistency that may not be maintained by quasi serializable executions.

2 Preliminaries

An HDDBS consists of a set $\mathcal{D}$[1] of data items and a set $\mathcal{T}$ of transactions. The data item set $\mathcal{D}$ consists of n subsets, $\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_n$, called local databases[2]. The transaction set $\mathcal{T}$ consists of $n+1$ subsets, $\mathcal{G}, \mathcal{L}_1, \mathcal{L}_2, ..., \mathcal{L}_n$, where $\mathcal{L}_i$ is a set of local transactions that access $\mathcal{D}_i$ only, while $\mathcal{G}$ is a set of global transactions that access more than one local database. A global transaction G_i consists of a set of subtransactions $\{G_{i,1}, G_{i,2}, ..., G_{i,n}\}$, where the subtransaction $G_{i,j}$ accesses $\mathcal{D}_j$ only. The data item set $\mathcal{D}_i$, together with the transaction set $\mathcal{T}_i = \mathcal{L}_i \cup \mathcal{G}_i$ where $\mathcal{G}_i = \{G_{j,i} \mid G_j \in \mathcal{G}\}$, forms the local database system $LDBS_i$.

[1]In the paper, we use italic letters to denote instances, e.g., lower case for data items and upper case for transactions, calligraphic letters to denote sets, and roman letters to denote acronyms.

[2]We assume that local databases are disjoint. In other words, there is no replication at global level.

2.1 Notations

We review and introduce some of the basic concepts that will be useful throughout the paper.

Transactions and Value Dependency

A *transaction* T_i is a finite set of operations. Each operation is either a *read operation* reading a data item x, denoted $r_i(x)$, or a *write operation* writing a data item x, denoted $w_i(x)$. We let $\mathcal{R}(T_i)$ and $\mathcal{W}(T_i)$ be the sets of read and write operations of T_i, respectively and $\mathcal{O}(T_i) = \mathcal{R}(T_i) \cup \mathcal{W}(T_i)$ the set of all operations in T_i.

Operations in a simple transaction (i.e., a local transaction or a global subtransaction) are linearly ordered (*execution order*). We assume that if a transaction both reads and writes a data item. the read operation precedes the write operation in the execution order. Execution orders between operations of different subtransactions of the same global transaction, however, are not specified[3].

Operations in a transaction are also partially ordered according to their value dependency. *Value dependency* is a relation between a write operation and a read operation of the same transaction. More specifically, a write operation depends on a read operation if the value it writes is a function of the value read by the read operation. We assume that there is value dependency between each write operation and a read operation of the same simple transaction which precedes it in the execution order. A write operation in a subtransaction may also depend on a read operation in another subtransaction of the same global transaction. This kind of remote value dependency must be explicitly specified in order to execute transactions correctly.

Definition 2.1 (Simple transactions) *A simple transaction T is a pair $< \mathcal{O}(T), \prec_{eo}^{T} >$, where $O(T)$ is the set of operations of T and $\prec_{eo}^{T}$ is a linear order (execution order) in which operations in $O(T)$ are executed.*

Given a simple transaction T, its value dependency is formally defined as,

$\prec_{vd}^{T} = \{(o_i, o_j) \mid o_i \in \mathcal{R}(T), o_j \in \mathcal{W}(T) \text{ and } o_i \prec_{eo}^{T} o_j\}$

Definition 2.2 (Global transactions) *A global transaction G_0 is a pair $< \mathcal{TS}(G_0), \prec_{rvd}^{G_0} >$, where $\mathcal{TS}(G_0)$ is a set of simple transactions (called global subtransactions) and $\prec_{rvd}^{G_0}$ is a binary relation over $\mathcal{O}(G_0) = \cup_{T \in \mathcal{TS}(G_0)} \mathcal{O}(T)$ representing the remote value dependency of G_0,*

[3] Specification and coordination of execution order of operations of different subtransactions are discussed in [LER89].

$$\prec_{rvd}^{G_0} = \{(o_i, o_j) \mid \exists x, y \in \mathcal{D} \text{ and } G_{0,i}, G_{0,j} \in \mathcal{TS}(G_0) \text{ such that } o_i = r_i(x) \in \mathcal{R}(G_{0,i}),$$
$$o_j = w_j(y) \in \mathcal{W}(G_{0,j}) \text{ and } y = f(x) \text{ for some function } f\}.$$

Remote value dependency is included in the definition of global transactions because it is both necessary for execution of transactions and useful in maintaining transaction consistency of HDDBSs (see section 4).

Given a global transaction G, we define $\prec_{eo}^{G} = \cup_{T \in \mathcal{TS}(G)} \prec_{eo}^{T}$ and $\prec_{vd}^{G} = \prec_{rvd}^{G} \bigcup (\cup_{T \in \mathcal{TS}(G)} \prec_{vd}^{T})$ to be its execution order and value dependency, respectively.

Executions and Transaction Interference

Definition 2.3 (Local executions) *A local execution E_l in $LDBS_l$ is an interleaved sequence of operations of transactions in $\mathcal{T}_l$, with the following property: for $o_i, o_j \in \mathcal{O}(T)$ where $T \in \mathcal{T}_l$, if $o_i \prec_{eo}^{T} o_j$ then o_i precedes o_j in E_l.*

We use $\prec_{eo}^{E_l}$ to denote the execution order of operations in local execution E_l.

$\prec_{eo}^{E_l} = \{(o_i, o_j) \mid \exists T_i, T_j \in \mathcal{T}_l \text{ such that } o_i \in \mathcal{O}(T_i), o_j \in \mathcal{O}(T_j) \text{ and } o_i \text{ precedes } o_j \text{ in } E_l\}$

Definition 2.4 (Global executions) *A global execution E in an HDDBS consists of a set of local executions, $E = \{E_1, E_2, ..., E_n\}$, where E_l is the local execution at $LDBS_l$.*

The execution order of a global execution E is defined as $\prec_{eo}^{E} = \cup_{l=1}^{n} \prec_{eo}^{E_l}$.

One way for a transaction to be influenced by other transactions is to read values they wrote, as defined in the following *read from* relation.

$$\prec_{rf}^{E} = \{(o_i, o_j) \mid \exists x \in \mathcal{D} \text{ such that } o_i = w_i(x), o_j = r_j(x) \text{ and } o_i \prec_{eo}^{E} o_j$$
$$\text{and } \nexists o_k = w_k(x) \text{ such that } o_i \prec_{eo}^{E} o_k \prec_{eo}^{E} o_j\}.$$

We say that a transaction T_j *indirectly reads from* transaction T_i in E if there exist $o_i \in \mathcal{W}(T_i)$ and $o_j \in \mathcal{R}(T_j)$ such that $(o_i, o_j) \in (\prec_{rf}^{E} \bigcup (\cup_{T \in \mathcal{T}} \prec_{vd}^{T}))^*$. Similarly, we say that a transaction T_j *indirectly depends on* transaction T_i in E if there exist $o_i \in \mathcal{R}(T_i)$ and $o_j \in \mathcal{W}(T_j)$ such that $(o_i, o_j) \in (\prec_{rf}^{E} \bigcup (\cup_{T \in \mathcal{T}} \prec_{vd}^{T}))^*$.

Another form of transaction interference is *over writing*.

$$\prec_{ow}^{E} = \{(o_i, o_j) \mid \exists x \in \mathcal{D} \text{ such that } o_i = w_i(x), o_j = w_j(x) \text{ and } o_i \prec_{eo}^{E} o_j$$
$$\text{and } \nexists o_k = w_k(x) \text{ such that } o_i \prec_{eo}^{E} o_k \prec_{eo}^{E} o_j\}.$$

Read from relation, together with over write relation, defines the interference among transactions in an execution.

$$\prec_{ti}^{E} = \{(T_i, T_j) \mid \exists o_i \in \mathcal{W}(T_i), o_j \in \mathcal{O}(T_j) \text{ such that either } (o_i, o_j) \in \prec_{ow}^{E} \text{ or } (o_j, o_i) \in (\prec_{rf}^{E} \bigcup (\cup_{T \in \mathcal{T}} \prec_{vd}^{T}))^{*}\}.$$

In this paper, we distinguish three kinds of transaction interference.

- **Local interference:** between transactions executed at the same site.
  $\prec_{li}^{E} = \{(T_i, T_j) \in \prec_{ti}^{E} \mid \exists l \text{ such that } T_i, T_j \in \mathcal{T}_l\}$
- **Global interference:** between global transactions.
  $\prec_{gi}^{E} = \{(T_i, T_j) \in \prec_{ti}^{E} \mid T_i, T_j \in \mathcal{G}\}$
- **Distributed interference:** between local transactions executed at different sites.
  $\prec_{di}^{E} = \{(T_i, T_j) \in \prec_{ti}^{E} \mid \exists l_i, l_j \text{ such that } T_i \in \mathcal{L}_{l_i}, T_j \in \mathcal{L}_{l_j} \text{ and } l_i \neq l_j\}$

Clearly, $\prec_{ti}^{E} = \prec_{li}^{E} \cup \prec_{gi}^{E} \cup \prec_{di}^{E}$. In addition, we have the following theorem.

Theorem 2.1 *$\prec_{ti}^{E}$ is acyclic if and only if $\prec_{li}^{E}$, $\prec_{gi}^{E}$ and $\prec_{di}^{E}$ are all acyclic.*

2.2 Example – International Banking

The HDDBS of an international bank federation consists of local databases of member banks at each country. Each local database consists of individual accounts. A customer may have accounts at one or more banks and manipulates his accounts in the usual way. In particular, he can either deposit money to or check balance of accounts at one or more banks at a time. He can also transfer money from an account at one bank to those at other banks.

Example 2.1 *Suppose that a user wants to transfer a certain amount of money from one of his account x at bank A to another account y at bank B. The request can be expressed as follows.*

begin_request T
  read$(x, balance_1)$;
  write$(x, balance_1 - amount)$;
  read$(y, balance_2)$;
  write$(y, balance_2 + amount)$;
end_request T

T is decomposed into the following two subrequests.

begin_subrequest T_A

  read$(x, balance_1)$;

  send$(B, amount)$;

  write$(x, balance_1 - amount)$;

end_subrequest T_A

begin_subrequest T_B

  read$(y, balance_2)$;

  receive$(A, amount)$;

  write$(y, balance_2 + amount)$;

end_subrequest T_B

There is value dependency between the read operation in T_A and the write operation in T_B.

The subrequests T_A and T_B can be expressed as simple transactions as follows.

$T_A = < O(T_A), \prec_{co}^{T_A} >$, *where* $O(T_A) = \{r_A(x), w_A(x)\}$ *and* $\prec_{co}^{T_A} = \{(r_A(x), w_A(x))\}$.

$T_B = < O(T_B), \prec_{co}^{T_B} >$, *where* $O(T_B) = \{r_B(y), w_B(y)\}$ *and* $\prec_{co}^{T_B} = \{(r_B(y), w_B(y))\}$.

And T can be expressed as a global transaction.

$T = < TS(T), \prec_{rvd}^{T} >$ *where* $TS(T) = \{T_A, T_B\}$ *and* $\prec_{rvd}^{T} = \{(r_A(x), w_B(y))\}$.

In cases where value dependency is not important, a global transaction can be simply expressed as a set of subtransactions. For example,

$T = \{T_A, T_B\}$, *where* $T_A : r_A(x)w_A(x)$ *and* $T_B : r_B(y)w_B(y)$.

3 Transaction Consistency of HDDBSs

In this section, we study the appropriateness of quasi serializability with respect to transaction consistency. An execution maintains transaction consistency if all transactions in the execution interfere with each other properly (e.g., in a partial order). We show, in this section, that quasi serializable executions maintain consistency for global transactions, as well as consistency for the transactions that appear in the same local execution. In the next section, we show that, under certain restrictions on remote value dependency of global transactions, quasi serializable executions maintain consistency for local transactions that appear in different local executions. Techniques and mechanisms that control remote value dependency will also be discussed in the next section.

We assume that initial HDDBS states are consistent with respect to any type of transaction consistent.

Definition 3.1 (T-Consistency of HDDBSs) *An HDDBS state is T-consistent if it has resulted from the initial state by an execution E in which*

1. *Local executions are serializable.*
2. *$\prec_{li}^{E}$ is acyclic.*
3. *If $(T_i, T_j) \in \prec_{rf}^{E}$, then $\forall x \in \mathcal{W}(T_i) \cap \mathcal{R}(T_j)$, T_j reads x from T_i.*

An execution maintains T-consistency of an HDDBS if it maintains conventional transaction consistency of all LDBSs. In addition, all transactions interfere with each other in a partial order (condition 2) and if a transaction is affected by (i.e., reads from) another transaction, the influence is complete (condition 3). Therefore, anomalies like inconsistent retrieval will not occur.

To study the ability of quasi serializable executions to maintain T-consistency, let us introduce more notations. We say that an HDDBS state is L-consistent (or G-consistent, D-consistent) if it is resulted from an initial state by an execution E in which

1. Local executions are serializable.
2. $\prec_{li}^{E}$ ($\prec_{gi}^{E}$, $\prec_{di}^{E}$) is acyclic.
3. $\forall T_i, T_j \in \mathcal{T}_l$ (or $\mathcal{G}, \mathcal{L}$), if $(T_i, T_j) \in \prec_{rf}^{E}$, then T_j reads x from T_i for all $x \in \mathcal{W}(T_i) \cap \mathcal{R}(T_j)$.

Theorem 3.1 *An HDDBS state is T-consistent if and only if it is L-consistent, G-consistent and D-consistent.*

Proof: (*if*) Given an HDDBS state which is resulted from an initial state by an execution E. Suppose that it is not T-consistent. There are three cases.

Case 1. $\exists l$ such that E_l is not serializable. Then, the state is not L-consistent.

Case 2. $\prec_{li}^{E}$ is cyclic. According to theorem 2.1, either $\prec_{li}^{E}$ or $\prec_{gi}^{E}$ or $\prec_{di}^{E}$ is also cyclic.

Case 3. $\exists T_i, T_j$ and $x \in \mathcal{R}(T_j) \cap \mathcal{W}(T_i)$, such that $(T_i, T_j) \in \prec_{rf} E$. T_j does not read x from T_i. Let us consider the following three cases.

- $\exists l$ such that $T_i, T_j \in \mathcal{T}_l$: The state is not L-consistent.
- $T_i, T_j \in \mathcal{G}$: The state is not G-consistent.
- $T_i, T_j \in \mathcal{L}$: The state is not D-consistent.

It is not hard to see that the above three cases exhaust all possibilities. □

Clearly, all serializable executions maintain T-consistency of HDDBSs. Quasi serializable executions maintain L-consistency and G-consistency. They also satisfy the first and third conditions of T-consistency, as the following theorem shows.

Theorem 3.2 *Given a quasi serializable execution E. If $(T_i, T_j) \in \prec_{rf}^{E}$, then $\forall x \in \mathcal{W}(T_i) \cap \mathcal{R}(T_j)$, T_j reads x from T_i.*

Proof: Given an execution E and two transactions T_i and T_j such that $(T_i, T_j) \in \prec_{rf}^{E}$. Let us consider the following three cases.

- $\exists l$ such that $T_i, T_j \in \mathcal{T}_l$. The theorem holds because the local execution is serializable.
- $T_i, T_j \in \mathcal{G}$. The theorem also holds because global transactions are executed sequentially in quasi serial executions.
- Otherwise. Impossible. □

Therefore, a quasi serializable execution E maintains T-consistency if $\prec_{li}^{E}$, $\prec_{gi}^{E}$ and $\prec_{di}^{E}$ are all acyclic.

Theorem 3.3 *A quasi serializable execution maintains L-consistency and G-consistency of HDDBSs. However, it may violate D-consistency of HDDBSs.*

Proof: Given a quasi serializable execution E. It maintains L-consistency (i.e., $\prec_{li}^{E}$ is acyclic) because local executions are serializable. $\prec_{gi}^{E}$ is also acyclic because global transactions are executed sequentially in the quasi serial execution. $\prec_{di}^{E}$, however, may be cyclic, as the following example shows. □

Example 3.1 (International banking) *Consider bank A with accounts a and b, and bank B with accounts x and y. Let G_1 and G_2 be two global transactions, L_1, L_2 be two local transactions issued at A and B, respectively.*

$G_1 = < \{G_{1,1}, G_{1,2}\}, \prec_{rvd}^{G_1} >$, *where* $G_{1,1} : r_{g_1}(a)w_{g_1}(a), G_{1,2} : r_{g_1}(x)w_{g_1}(x)$ *and* $\prec_{rvd}^{G_1} = \{(r_{g_1}(a), w_{g_1}(x))\}$.

$G_2 = < \{G_{2,1}, G_{2,2}\}, \prec_{rvd}^{G_2} >$, *where* $G_{1,1} : r_{g_2}(y)w_{g_2}(y), G_{2,2} : r_{g_2}(b)w_{g_2}(b)$ *and* $\prec_{rvd}^{G_1} = \{(r_{g_2}(y), w_{g_2}(b))\}$.

$L_1 : r_{l_1}(a)w_{l_1}(a)r_{l_1}(b)$

$L_2 : r_{l_2}(x)r_{l_2}(y)w_{l_2}(y)$

Let E be an execution of $\{G_1, G_2, L_1, L_2\}$.

$E = \{E_1, E_2\}$, *where*

$E_1 : r_{l_1}(a)w_{l_1}(a)r_{g_1}(a)w_{g_1}(a)r_{g_2}(b)w_{g_2}(b)r_{l_1}(b)$

$E_2 : r_{g_1}(x)w_{g_1}(x)r_{l_2}(x)r_{l_2}(y)w_{l_2}(y)r_{g_2}(y)w_{g_2}(y)$

Then, E is quasi serializable. However, $\prec_{li}^{E}$ is cyclic because $(T_1, T_2), (T_2, T_1) \in \prec_{li}^{E}$.

Suppose that G_1 transfers money from a to x, G_2 transfers money from y to b, L_1 reads balance of $a + b$ and deposits money to a, and L_2 reads balance of $x + y$ and deposits money to y. The balance L_1 reads includes that deposited by L_2 and the balance L_2 reads also includes that deposited by L_1. Therefore, L_1, L_2 affect each other mutually. □

In summary, a quasi serializable execution E maintains T-consistency of an HDDBS if and only if $\prec_{di}^{E}$ is acyclic.

4 Maintaining Transaction Consistency

There are two issues in maintaining transaction consistency using quasi serializable executions. The first is to guarantee the quasi serializability of executions, and the second is to guarantee the acyclicity of the distributed interference relation of an execution. We have discussed the first issue in [ED90] and [DE90]. In this section, we study the problem of maintaining acyclicity of distributed interference relations.

Distributed interference between local transactions at different sites is introduced by global transactions (via remote value dependency). Therefore, it can be prevented by imposing restrictions on remote value dependency of global transactions in the execution. To formulate such a restriction, let us first introduce the notion of value dependency graph of an execution.

Given an execution E of transactions $\mathcal{T}$ and a time t in its lifetime. Let δ_{max} be the maximum elapse time of a single local transaction and $\mathcal{G}(t)$ the set of global transactions that are active in $(t - \delta_{max}, t)$ in E.

Definition 4.1 (Value dependency graphs of executions) *The value dependency graph of execution E at time t, $VDG(E,t)$, is an undirected graph $< \mathcal{V}, \mathcal{A} >$, where $\mathcal{V} = \{LDBS_1, LDBS_2, ..., LDBS_n\}$ and $\mathcal{A} = \{(LDBS_i, LDBS_j) \mid \exists G_l \in \mathcal{G}(t), o_i \in \mathcal{R}(G_{l,i})$ and $o_j \in \mathcal{W}(G_{l,j})$ such that $(o_i, o_j) \in \prec_{rvd}^{G_l}\}$.*

Theorem 4.1 *Given an execution E. $\prec_{di}^{E}$ is acyclic if $VDG(E,t)$ is acyclic for all t in E's lifetime.*

Proof: Suppose that $\prec_{di}^{E}$ is cyclic: $\exists T_i \in \mathcal{L}_i$, $T_j \in \mathcal{L}_j (i \neq j)$ such that $(T_i, T_j) \in \prec_{di}^{E}$ and $(T_j, T_i) \in \prec_{di}^{E}$. Then, there exist $o_{p_1} \in \mathcal{R}(G_{p_1,i}), o_{p_2} \in \mathcal{W}(G_{p_2,i'}), ..., o_{p_{l-1}} \in \mathcal{R}(G_{p_{l-1},j'}), o_{p_l} \in \mathcal{W}(G_{p_l,j}) (l \geq 1)$, where $G_{p_1}, G_{p_2}, ..., G_{p_l} \in \mathcal{G}$, such that o_{p_l} indirectly depends on $o_{p_{l-1}}$, ..., o_{p_2} indirectly depends on o_{p_1}. Similarly, there exist $o_{q_1} \in \mathcal{W}(G_{q_1,i}), o_{q_2} \in \mathcal{R}(G_{q_2,i_2}), ..., o_{q_{m-1}} \in \mathcal{W}(G_{q_{m-1},j_{m-1}}), o_{q_m} \in \mathcal{R}(G_{q_m,j}) (m \geq 1)$, where $G_{q_1}, G_{q_2}, ..., G_{q_m} \in \mathcal{G}$, such that o_{q_1} indirectly depends on o_{q_2}, ..., $o_{q_{m-1}}$ indirectly depends on o_{q_m}. Therefore, there exist t_1, t_2 such that $T_i, G_{p_1}, G_{p_2}, ..., G_{p_l}$ and T_j are all active at time t_1, and $T_i, G_{q_1}, G_{q_2}, ..., G_{q_m}$ and T_j are all active at time t_2. Clearly, $t_1 \in (t_2, t_2 + \delta_{max})$ (assume that $t_1 < t_2$). In other words, $G_{p_1}, ..., G_{p_l}, G_{q_1}, ..., G_{q_m} \in \mathcal{G}(t_1)$. Therefore, $VDG(E, t_1)$ is cyclic. $\square$

Mechanisms based on theorem 4.1 can be constructed to maintain acyclicity of value dependency graphs of executions. In the mechanisms, a data structure is maintained to store the current value dependency graph of the execution as defined in definition 4.1. Every global transaction is checked against the graph before it is submitted. It is delayed if it creates cycles in the graph, and submitted otherwise. Edges of obsolete global transactions are purged δ_{max} later after its commitment. Delayed transactions may be retried each time some edges are purged.

Another way to maintain acyclicity of the value dependency graph of an execution is to take advantage of restrictions on site level information flow. For example, in some secure database environments, information is only allowed to flow from a site to those with higher (or the same) security classifications. In such environments, value dependency graph of an execution is always acyclic.

5 Conclusion

Quasi serializability is a new correctness criterion for concurrency control in HDDBSs. It is attractive in the HDDBS environment not only because it can be effectively maintained at global level without violating local autonomy, but also because it assures, with certain restrictions of (or control over) executions of global transactions, HDDBS consistency.

In this paper, we have studied appropriateness of quasi serializability with respect to transaction consistency in HDDBSs. The main results of the paper are: (1) identifying the aspects of transaction consistency that can be effectively maintained by quasi serializable executions and (2) proposing restrictions and techniques to prevent possible violation of those aspects of transaction consistency that may not be maintained by quasi serializable executions.

Another important issue of concurrency control using quasi serializable executions is preserving data integrity of HDDBSs. We are now working on this problem and the results will be reported elsewhere.

References

[DE89] W. Du and A. Elmagarmid. Quasi serializability: a correctness criterion for global concurrency control in InterBase. In *Proceedings of the International Conference on Very Large Data Bases*, pages 347–355, Amsterdam, The Netherlands, August 1989.

[DE90] W. DU and A. Elmagarmid. Maintaining quasi serializability in HDDBSs. Technical Report CSD-TR-971, Purdue University, March 1990.

[DEK90] W. Du, A. Elmagarmid, and W. Kim. Effects of local autonomy on heterogeneous distributed database systems. Technical Report ACT-OODS-EI-059-90, MCC, February 1990.

[DELO89] W. Du, A. Elmagarmid, Y. Leu, and S. Ostermann. Effects of autonomy on global concurrency control in heterogeneous distributed database systems. In *Proceedings of the Second International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, pages 113–120, Gaithersburg, Maryland, October 1989.

[ED90] A. Elmagarmid and W. Du. A paradigm for concurrency control in heterogeneous distributed database systems. In *Proceedings of the Sixth International Conference on Data Engineering*, Los Angeles, California, February 1990.

[LER89] Y. Leu, A. Elmagarmid, and M. Rusinkiewicz. An extended transaction model for multidatabase systems. Technical Report CSD-TR-925, Computer Sciences Department, Purdue University, October 1989.