1988

# Quasi Serializability: A Correctness Criterion for InterBase

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

Report Number:
88-836

# QUASI SERIALIZABILITY: A CORRECTNESS CRITERION FOR INTERBASE

**Ahmed Elmagarmid**

**Department of Computer Sciences**
**Purdue University**
**West Lafayette, IN 47907**

**CSD TR #88-836**
**December 1988**

**Abstract**

In this paper, we introduce Quasi Serializability, a correctness criterion for concurrency control in heterogeneous distributed database environments. A global history is quasi serializable if it is (conflict) equivalent to a quasi serial history in which global transactions are submitted serially. Quasi serializability theory is an extension of serializability. We study the relationships between serializability and quasi serializability and the reasons quasi serializability can be used as a correctness criterion in heterogeneous distributed database environments. We also use quasi serializability theory to give a correctness proof for an altruistic locking algorithm.

**Key words and phrases:** heterogeneous/federated database systems, concurrency control in heterogeneous distributed database systems, quasi serializability.

# 1 Introduction

The InterBase project in the Computer Science Department at Purdue University is an effort to investigate multidatabase management systems. The goal of this project is ultimately to build a Heterogeneous Distributed Database System (HDDBS) that supports atomic updates across multiple databases.

An HDDBS integrates pre-existing database systems to support global applications accessing more than one element database. An HDDBS is different from a homogeneous distributed database system in that the element databases are autonomous and possibly heterogeneous.

HDDBSs have become a very attractive research area recently. Within this research area, the transaction processing problem, especially that of concurrency control, has been getting considerable attention. Many global concurrency control algorithms (protocols) have appeared in the literature [GL84] [LEM88] [GP85] [Pu86] [BST87] [EH88] [BS88]. However, little attention has been put on developing theoretical tools for proving the correctness of these algorithms.

Serializability has been generally used as the correctness criterion for the proposed concurrency control strategies. Unfortunately, serializability does not work well in heterogeneous distributed database environments. In [DELO88], we discussed the difficulties of maintaining global serializability in heterogeneous distributed database environments. The reason, we believe, is that serializability was originally introduced for centralized database environments and therefore is centralized in nature. Global concurrency control in heterogeneous distributed database environments, on the other hand, is hierarchical in nature due to the autonomies of the element databases. As a result, some of the proposed algorithms violate local autonomies (e.g., Sugihara's distributed cycle detection algorithm [Sug87]), while some allow low degree of concurrency (e.g., Breitbart's site graph testing protocol [BS88]), and others fail to maintain global serializability (e.g., [AGS87], [Pu86], [EH88] and [LEM88]).

The hierarchical nature of global concurrency control in HDDBSs makes it very difficult to maintain global serializability. On the other hand, it relieves the

GCC from some responsibilities (e.g., the correctness of local histories). This suggests that the correctness criteria for global concurrency control in HDDBSs should be based primarily on the behavior of global applications, with proper consideration of the effects of local applications on global applications. In this paper, we define such a criterion called quasi serializability (QSR). QSR is used as the correctness criterion for global concurrency control in InterBase.

In section 2, we define the terminology and assumptions used in this paper. We then introduce, in section 3, QSR as a correctness criterion for global concurrency control in HDDBSs. In section 4, we give a correctness proof, using QSR theory, of Alonso's altruistic locking algorithm [AGS87]. The correctness of this algorithm cannot be shown using serializability. Some concluding remarks are given in section 6.

# 2 Preliminaries

In this section, we review some basic concepts used in our study of quasi serializability. Many of these concepts are from literature on centralized database systems, adapted to a heterogeneous distributed database system (e.g. [BG85]). We also give the assumptions on which our study of quasi serializability is based.

A *heterogeneous distributed database system* consists of a set $D = \{D_1, D_2, ..., D_m\}$ of local database systems (LDBSs), a set $G = \{G_1, G_2, ..., G_n\}$ of global transactions, and a set $L = \bigcup_{i=1}^{m} L_i$ of local transactions, where $L_i = \{L_{i,1}, L_{i,2}, ..., L_{i,j_i}\}$. A *local transaction* is a transaction that accesses only one LDBS. The local transaction set $L_i$ contains those local transactions that access local database $D_i$. A *global transaction* is a transaction that accesses more than one LDBS. The global transaction $G_i$ consists of a set of global subtransactions, $\{G_{i,1}, G_{i,2}, ..., G_{i,m}\}$, where the subtransaction $G_{i,j}$ access local database $D_j$.

A *global history* $H$ over $G \cup L$ in an HDDBS is a set of local histories $H = \{H_1, H_2, ..., H_m\}$, where the local history $H_i$ (at local database $D_i$) is defined over global subtransactions $G_{1,i}, G_{2,i}, ..., G_{n,i}$, and local transactions $L_{i,1}, ..., L_{i,j_i}$.

Concurrency control is a component of a database system that is concerned with deciding what actions should be taken in response to requests by individual transactions so that only correct histories are generated. In an HDDBS, concurrency control is performed hierarchically. Each of the LDBSs is autonomous. The local concurrency controller (LCC) at each LDBS is responsible for the correctness of the history at that site. The GCC, which is added on top of the existing LDBSs, coordinates the histories at all local sites to guarantee the correctness of the global history [GP86].

In this paper, LDBSs are assumed to be non-replicated and autonomous. Autonomy is defined as follows [EV87] [DELO88].

- *Design autonomy:* Each of the LDBS is free to use whatever data models and transaction management algorithms it wishes. Therefore, LDBSs might differ from each other in data models (network, hierarchical and relational) and transaction management strategies.

- *Communication autonomy:* Each of the LCCs is allowed to make independent decisions as to what information to provide to other LCCs or the GCC, and when to provide it. Therefore, the subtransactions of the same global transaction run independently at their local sites.

- *Execution autonomy:* Each of the LCCs is free to commit or restart any transactions running at its local site.

# 3 Quasi Serializability

In this section we introduce Quasi Serializability. We first give its definition, and then discuss its various properties as a correctness criterion for concurrency control. We also discuss the relationships between QSR and other correctness criteria.

We assume that the reader is familiar with the basic theory and notations of serializability (see, e.g., [BG85]).

## 3.1 Definitions

As mentioned earlier, QSR is used as a correctness criterion for global concurrency control. The basic idea is that, in order to preserve the global database consistency, global transactions should be executed in a serializable way, with proper consideration of the effects of local transactions. To understand this effect, let us expand the notion of conflict.

Let $o_i$ and $o_j$ be operations of two different transactions in a local history $H$, where $o_i$ precedes $o_j$. We then say that $o_i$ *directly conflicts* with $o_j$ in $H$ if they both access the same data item and at least one of them is a write operation. Whereas if $o_i$ and $o_j$ belong to the same local transaction (or global subtransaction), then we always say that $o_i$ directly conflict with $o_j$ in $H$. This, however, does not apply to operations of different subtransactions belonging to the same global transaction.

Let $o_i$ and $o_j$ be operations of two different transactions in a local history $H$. We say that $o_i$ *indirectly conflicts* with $o_j$ in $H$ if there exist operations $o_1$, $o_2$, ..., $o_k$ ($k \geq 1$) of other transactions such that $o_i$ directly conflicts with $o_1$ in $H$, $o_1$ directly conflicts with $o_2$ in $H$, ..., and $o_k$ directly conflicts with $o_j$ in $H$.

Let $G_i$ and $G_j$ be global transactions in a global history $H$. We say that $G_i$ *directly conflicts* with $G_j$ in $H$, denoted $G_i \rightarrow_d^H G_j$, if one of $G_i$'s operations directly conflicts with one of $G_j$'s operations in a local history of $H$. We say that $G_i$ *indirectly conflicts* with $G_j$ in $H$, denoted $G_i \rightarrow_i^H G_j$, if one of $G_i$'s operations indirectly conflicts with one of $G_j$'s operations in a local history $H$. We also simply say that $G_i$ conflicts with $G_j$ in $H$, denoted $G_i \rightarrow^H G_j$, if either $G_i \rightarrow_d^H G_j$ or $G_i \rightarrow_i^H G_j$.

Notice that the conflict relation we defined here is history dependent. In addition, it is irreflexive. Therefore, an operation, $o_i$, (directly or indirectly) conflicts with another operation, $o_j$, in a history only if $o_i$ precedes $o_j$ in that history. Notice also that a global transaction might indirectly conflict with another global transaction in a history even though they do not access any common data items. The indirect conflicts are introduced by other transactions.

It is the indirect conflicts that model the effect of local transactions on global transactions in a history.

Now let us introduce the notion of a *quasi serial history*. Unlike a serial history, only global transactions are required to execute serially in a quasi serial history. As we shall see later, this, together with the serializability of local histories, is sufficient to guarantee the correctness of global concurrency control in heterogeneous distributed database environments.

**Definition 3.1 (Quasi Serial History)** *A global history is quasi serial if*

1. *all local histories are (conflict) serializable; and*
2. *there exists a total order of all global transactions such that for every two global transactions $G_i$ and $G_j$ where $G_i$ precedes $G_j$ in the order, all $G_i$'s operations precede $G_j$'s operations in all local histories in which they both appear.*

Two global histories of an HDDBS are (conflict) *equivalent*, denoted $\equiv$, if their corresponding local histories are all (conflict) equivalent [BG85].

**Definition 3.2 (Quasi Serializability)** *A history is quasi serializable if it is (conflict) equivalent to a quasi serial history.*

In a quasi serializable history, all local histories are serializable. In addition, global transactions are executed in a way that is serializable in terms of both direct and indirect conflicts. This kind of serializability is achieved by taking into account conflicts between both local and global transactions, although we are only interested in the behavior of global transactions.

Example 3.1 Consider an HDDBS consisting of two local databases, $\text{LDBS}_1$ and $\text{LDBS}_2$, where data items $a$ and $b$ are at $\text{LDBS}_1$, and $c$, $d$ and $e$ are at $\text{LDBS}_2$. The following global transactions are submitted to the HDDBS:

$G_1 : w_{g_1}(a) r_{g_1}(d)$

$G_2 : r_{g_2}(b) r_{g_2}(c) w_{g_2}(e)$

Let $L_1$ and $L_2$ be the local transactions submitted at $\text{LDBS}_1$ and $\text{LDBS}_2$, respectively:

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

$L_2 : w_{l_2}(d)r_{l_2}(e)$

Let $H_1$ and $H_2$ be local histories at $\text{LDBS}_1$ and $\text{LDBS}_2$, respectively:

$H_1 : w_{g_1}(a)r_{l_1}(a)w_{l_1}(b)r_{g_2}(b)$

$H_2 : r_{g_2}(c)w_{l_2}(d)r_{g_1}(d)w_{g_2}(e)r_{l_2}(e)$

Let $H = \{H_1, H_2\}$. Then $H$ is quasi serializable. It is equivalent to the quasi serial history $H' = \{H'_1, H'_2\}$, where

$H'_1 : w_{g_1}(a)r_{l_1}(a)w_{l_1}(b)r_{g_2}(b)$

$H'_2 : w_{l_2}(d)r_{g_1}(d)r_{g_2}(c)w_{g_2}(e)r_{l_2}(e)$ □

## 3.2 The Quasi Serializability Theorem

There is a convenient graph-theoretic characterization of quasi serializability which is described in the following theorem. Let us first introduce the Quasi Serialization Graph (QSG).

The *quasi serialization graph* for a global history $H$, denoted QSG($H$), is a directed graph whose nodes are the global transactions that are committed in $H$, and whose edges are all the relations $G_i \rightarrow G_j$ $(i \neq j)$ such that $G_i \rightarrow^H G_j$.

**Theorem 3.1 (The Quasi Serializability Theorem)** *A global history $H$ is quasi serializable if and only if all local histories are (conflict) serializable and QSG($H$) is acyclic.*

*Proof:* (if) Suppose $H = \{H_1, H_2, ..., H_m\}$ is a global history over $G \cup L$, where $G$ is a set of global transactions and $L$ is a set of local transactions. Without loss of generality, we assume that $G_1, G_2, ..., G_n$ are the global transactions that are committed in $H$ and, therefore, are the nodes of QSG($H$). Since QSG($H$) is acyclic, it may be topologically sorted. Let $i_1, i_2, ..., i_n$ be a permutation of $1, 2, ..., n$ such that $G_{i_1}, G_{i_2}, ..., G_{i_n}$ is a topological sort of QSG($H$). For each local history $H_j$ $(1 \leq j \leq m)$, assume that $G_{i_{j_1},j}, G_{i_{j_2},j}, ..., G_{i_{j_l},j}$ are the global subtransactions that appear in $H_j$ $(1 \leq j_1 \leq j_2 \leq ... \leq j_l \leq n)$. We show, below, that there is another serializable local history $H'_j$, equivalent to $H_j$, such

that all of $G_{i_{j_1},j}$'s operations precede $G_{i_{j_2},j}$'s operations in $H_j'$, all of $G_{i_{j_2},j}$'s operations precede $G_{i_{j_3}}$'s operations in $H_j'$, and so on.

Let $OP_{j_1}$ be the set of all operations in $G_{i_{j_1},j}$ and those operations in $H_j$ that conflict with and precede one of $G_{i_{j_1},j}$'s operations. Let $OP_{j_2}$ be the set of all operations of $G_{i_{j_2},j}$ and those operations in $H_j$ that conflict with and precede one of $G_{i_{j_2},j}$'s operations and are not in $OP_{j_1}$. In general, $OP_{j_k}$ ($1 \leq k \leq l$) is the set of all operations in $G_{i_{j_k},j}$ and those operations that conflict with and precede one of $G_{i_{j_k},j}$'s operations and are not in $OP_{j_1} \cup OP_{j_2} \cup ... \cup OP_{j_{k-1}}$. $OP_{j_{l+1}}$ is the set of all other operations in $H_j$. For any two integers $p$ and $q$ ($1 \leq p < q \leq l+1$), $OP_{j_p}$ and $OP_{j_q}$ are disjoint and no operations in $OP_{j_q}$ both conflict with and precede any operations in $OP_{j_p}$ in $H_j$ because none of $G_{i_{j_q},j}$'s operations both conflict with and precede any of $G_{i_{j_p},j}$'s operations in $H_j$. The $H_j'$ can be constructed from $H_j$ as follows. It is made up of all operations in $OP_{j_1}$, followed by all operations in $OP_{j_2}$, ..., followed by all operations in $OP_{j_l}$, and finally followed by all operations in $OP_{j_{l+1}}$. The orders of the operations in $OP_k$ ($1 \leq k \leq l+1$) are the same as in $H_j$. $H_j \equiv H_j'$ because $H_j'$ orders the conflicting operations in the same way the $H_j$ does. Since $H_j$ is serializable, $H_j'$ is also serializable. Let $H' = \{H_1', H_2', ..., H_m'\}$, then $H'$ is quasi serial and $H \equiv H'$. Therefore, $H$ is quasi serializable.

(only if) Suppose a global history $H$ is quasi serializable. Again, we assume that $G_1, G_2, ..., G_n$ are the global transactions that are committed in $H$ and let C($H$) be the committed projection of $H$. Let $H_s$ be the quasi serial history that is equivalent to C($H$). Suppose QSG($H$) is cyclic. Let $G_{i_1} \rightarrow G_{i_2} \rightarrow G_{i_3} ... \rightarrow G_{i_k} \rightarrow G_{i_1}$ be a cycle in QSG($H$), and therefore in QSG($H_s$). Let $G_{i_p}$ and $G_{i_q}$ be two global transactions that are in the cycle ($1 \leq p < q \leq k$). This means that some of $G_{i_p}$'s operations conflict with $G_{i_q}$'s operations, and some of $G_{i_q}$'s operations conflict with $G_{i_p}$'s operations. However, this is impossible because $H_s$ is quasi serial. (Recall that a global transaction conflicts with another global transaction only if some operations of the former precede some operations of the latter in a local history.) In other words, either all of $G_{i_p}$'s operations precede $G_{i_q}$'s operations in all local histories and therefore $G_{i_p} \rightarrow^H G_{i_q}$, or

all $G_{i_q}$'s operations precede $G_{i_p}$'s operations in all local histories and therefore $G_{i_q} \rightarrow^H G_{i_p}$. □

**Example 3.2** The QSG of the global history $H$ in example 3.1, as shown in Figure 1.(b), is acyclic. However, its serialization graph, as shown in Figure 1.(a), is cyclic. □

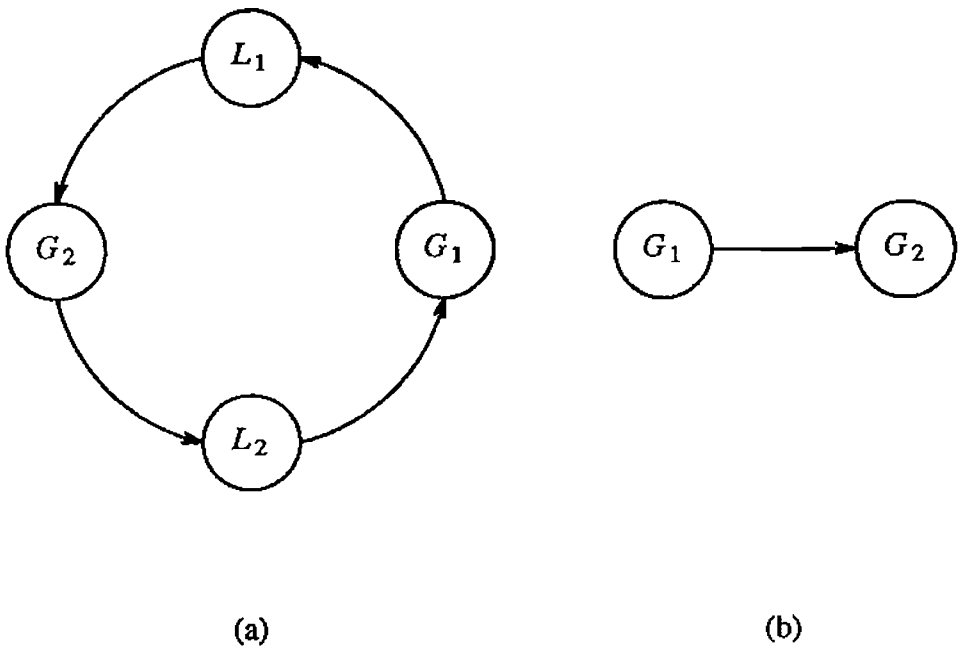


Figure 1: Serialization graph (a) and Quasi Serialization Graph (b) of $H$.

## 3.3 QSR as a Correctness Criterion for Concurrency Control

In this section, we show that quasi serializable histories are correct in terms of global concurrency control in heterogeneous distributed database environments. We do so by first discussing the database consistency problem, especially that of HDDBSs. We then discuss the ways that quasi serializable histories maintain the HDDBS consistency. Since our HDDBS model is non-replicated, we will not discuss the mutual consistency problem.

### 3.3.1 HDDBS Consistency Problem

It is generally accepted (see, e.g., [SIR76] [RSI78] and [SK80]) that a history is correct (or the database consistency is ensured) if

- Each transaction sees a consistent database.
- Each transaction eventually terminates.
- The final database after all transactions terminate is consistent.

Generally, a database is said to be consistent if it satisfies a set of integrity constraints. There are two types of integrity constraints associated with a database system:

1. Constraints on data items

   Data item constraints specify the real world restrictions on the values of data items. For example, the salary of an employee in a departmental database must always be greater than zero. These constraints are usually database dependent, but application independent. Therefore, they can be checked statically. In other words, whether a database satisfies the constraints can be checked entirely based on the values of data items.

2. Constraints on transactions

   Transaction constraints specify the general restrictions on the interference between transactions. For example, two transactions should not mutually influence each other. These constraints can be database independent, but they cannot be checked statically. In other words, whether a database satisfies the constraints depends on all transactions executed and the interference among them.

Notice that the constraints on transactions usually cannot be substituted for constraints on data items. To see this, let us consider the famous lost update anomaly (see, e.g. [BG85]). The lost update phenomenon occurs whenever two transactions, while attempting to modify a data item, both read the item's old value before either of them writes the item's new value. Both transactions

see the same (initial) database, which satisfies all constraints on data items. Although, the final database (which is the same as if only one transaction has been executed) also satisfies all constraints on data items, the execution is not correct.

Constraints on data items can be expressed as a set of predicates, however they are not usually explicitly given. One way of preserving these constraints is to execute the involved transactions sequentially.

Constraints on transactions, on the other hand, usually cannot be expressed in predicate form. As with data item constraints, they are usually not explicitly given. They are generally required because of the conflict between user transparency and concurrent execution of transactions. Again, serial execution of the involved transactions is sufficient to preserve these constraints.

In an HDDBS, each local database has its own set of integrity constraints. When the local databases are integrated into an HDDBS, these integrity constraints are combined, together with some integrity constraints for the global database, to form the integrity constraints of the HDDBS.

Since there is no replication among local databases, the constraints on data items of different local databases are defined on separate sets of data items. Therefore, there are no global constraints (with respect to the global database scheme) on data items at different local sites. This is for the following reasons. First, since each local database is designed independently, there are no relationships between data items at different databases. Therefore, it does not make any sense to impose restrictions on data items at different databases. Second, neither the users nor the LCCs of each local database are aware of the integration process (and therefore the global constraints). Therefore local transactions, even when they are executed alone, may not preserve these global constraints [1].

On the other hand, there might be global constraints imposed on the transactions. This is possible because these constraints can be totally independent of any local transactions (e.g., constraints between two global transactions). This

[1] Some people, however, believe that there should be some global constraints on data items which are not identical to the local constraints (see, e.g., [GP86]). However, they do not mention how these global constraints can be defined and preserved.

is also necessary, as the following example shows.

**Example 3.3** Consider an HDBS consisting of two local databases, $LDBS_1$ and $LDBS_2$, where data item $a$ is at $LDBS_1$ and data items $b$ and $c$ are at $LDBS_2$. The following global transactions are submitted to the HDBS:

$G_1 : w_{g_1}(a) r_{g_1}(c)$

$G_2 : r_{g_2}(a) w_{g_2}(b)$

Let $L$ be the local transaction submitted at $LDBS_1$:

$L : r_l(b) w_l(c)$

Let $H_1$ and $H_2$ be local histories at $LDBS_1$ and $LDBS_2$, respectively:

$H_1 : w_{g_1}(a) r_{g_2}(a)$

$H_2 : w_{g_2}(b) r_l(b) w_l(c) r_{g_1}(c)$

In $H_1$, $G_2$ reads the value, $a$, written by $G_1$ directly. In $H_2$, however, $G_1$ might read the value, $b$, written by $G_2$ indirectly (e.g., local transaction $L$ copies the value of $b$ to $c$). Therefore, each global transaction sees only part of the effect of the other. Obviously, this kind of global inconsistent retrieval should not be allowed. However, it cannot be detected by LCCs. □

In heterogeneous distributed database environments, a global database is consistent if and only if it satisfies the global constraints on transactions and all the local databases are consistent. Similarly, a global history preserves global database consistency if and only if it preserves the global constraints on transactions and all local histories preserve local database consistency.

### 3.3.2 Correctness of Quasi Serializable Histories

Since a quasi serializable history is equivalent to a quasi serial history, we only need to show the correctness of quasi serial histories. In other words, we need to show that any quasi serial history preserves the global constraints on transactions. This is true because of the serializability of local histories, and the serial execution of global transactions. To see this, let us investigate various kinds of constraints on transactions in heterogeneous distributed database environments:

- Constraints on local transactions (or global subtransactions) at the same LDBS: They are preserved because of the serializability of local histories.
- Constraints on global transactions: They are preserved because the global transactions executed serially.
- Constraints on local transactions (or global subtransactions) at different LDBSs: There is no direct relation between local transactions at different LDBSs, but there might be indirect relations between them (through global transactions). However, this is true only if there exist relations between subtransactions of the same global transaction. Since we have assumed that there is no communication between subtransactions of the same global transaction, there are no global constraints on local transactions (or global subtransactions) at different LDBSs.

In a quasi serializable history, each global transaction sees a consistent global database because the global transactions preceding it are executed serially. The final global database is also consistent because all global transactions are executed serially. Therefore, all quasi serializable histories are correct in heterogeneous distributed database environments.

**Example 3.4** Let us consider the global history $H$ in example 3.1. Since there is no communication between the two subtransactions of transaction $G_2$, the value of data item $e$ written by $G_2$ at $\text{LDBS}_2$ is not related to the value of data item $b$ read by $G_2$ at $\text{LDBS}_1$. Therefore, the value of data item $e$ read by local transaction $L_2$ at $\text{LDBS}_2$ is not related to the value of data item $b$ written by local transaction $L_1$ at $\text{LDBS}_1$. In other words, there is no relation between $L_1$ and $L_2$ (or they do not influence each other). The global constraints which can only be defined on global transactions are preserved because the global transactions are executed serially. □

It is worth noting that the above arguments are only true in heterogeneous distributed database environments. They may not hold in, for example, homogeneous distributed database environments. There are two reasons for this. First, in homogeneous environments, global constraints can be defined on data

items at different local databases. A quasi serializable history might not preserve these constraints. Second, it is possible for subtransactions of the same global transaction to communicate with each other in homogeneous environments. Therefore, it is also possible to define global constraints on transactions at different local databases. Again, a quasi serializable history might not preserve these constraints.

In summary, a quasi serializable history will preserve the global HDDBS consistency (and therefore is correct) if there is no replication among LDBSs and there is no communication between subtransactions of the same global transaction.

## 3.4 Relationships to Other Criteria

We now discuss the relationships between QSR and other criteria. Since both are defined on different database models, their meaning of correctness is also different. The results presented below only show the relationships in term of inclusiveness.

The most commonly used correctness criterion in general database environments (and also in heterogeneous distributed database environments) is serializability (see, e.g., [BS88]). Let $G$ be a set of global transactions and $L = \{L_1, ..., L_m\}$ be sets of local transactions at various local sites. A global history over $G \cup L$ is serializable if it is computationally equivalent to a serial global history over $G \cup L$ [2].

Two types of serializability exist, conflict serializability and view serializability. A global history over $G \cup L$ is conflict (or view) serializable if it is conflict (or view) equivalent to a serial history over $G \cup L$. In this section, we use CSR to denote the set of global histories which are conflict serializable, and VSR for those which are view serializable. We use QSR to denote the set of global histories that are quasi serializable.

For a global history, its quasi serialization graph is a subgraph of its serialization graph. Therefore, a conflict serializable global history is also quasi

[2] $L$ is treated as a set of "one site global transactions".

serializable. In other words, $CSR \subseteq QSR$.

**Theorem 3.2** *$CSR \subset QSR$.*

*Proof:* We only need to show that $CSR \neq QSR$. To see this, let us consider, for example, the global history $H$ in example 3.1. It is quasi serializable, but not serializable. □

However, this is not true for VSR because the serialization graph for a view serializable history may not be acyclic. To see this, let us consider the following example.

**Example 3.5** Consider an HDDBS consisting of two local databases, $LDBS_1$ and $LDBS_2$, where data items $a$, $b$ and $c$ are at $LDBS_1$ and $d$ is at $LDBS_2$. The following global transaction is submitted to the HDDBS:

$G_1 : w_{g_1}(a)w_{g_1}(b)w_{g_1}(c)r_{g_1}(d)$

Let $L_1$ and $L_2$ be two local transactions submitted at $LDBS_1$.

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

$L_2 : w_{l_2}(a)w_{l_2}(b)$

Let $H_1$ and $H_2$ be local histories at $LDBS_1$ and $LDBS_2$, respectively:

$H_1 : w_{g_1}(a)w_{l_1}(a)w_{l_1}(b)w_{g_1}(b)w_{l_2}(a)w_{l_2}(b)w_{g_1}(c)$

$H_2 : r_{g_1}(d)$

Let $H = \{H_1, H_2\}$. Then $H$ is view serializable. Since $H_1$ is not conflict serializable (see [BG85]), $H$ is not quasi serializable. □

**Theorem 3.3** *$VSR \not\subset QSR$.*

The reverse is also not true. For example, the global history, $H$, in example 3.1 is quasi serializable but it is not view serializable.

**Theorem 3.4** *$QSR \not\subset VSR$.*

The relationships among QSR, CSR and VSR are illustrated in Figure 2.

In [KS88], Korth proposed the use of predicatewise serializability as the correctness criterion for concurrency control in computer-aided design and office information systems. The basic idea of predicatewise serializability is that if the

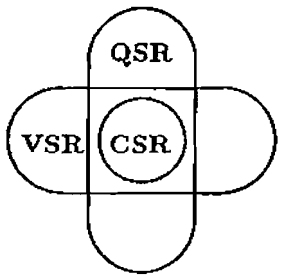


Figure 2: Relationships among QSR, CSR and VSR.

database consistency constraint is in conjunctive normal form, we can maintain the consistency constraint by enforcing serializability only with respect to data items which share a disjunctive clause. Clearly, predicatewise serializability concerns constraints on data items only. In heterogeneous distributed database environments, only the data items at the same LDBS can share a constraint clause. Therefore, a global history is predicatewise serializable if all the local histories are serializable. Since all the local histories in a quasi serializable history are serializable, each quasi serializable history is also predicatewise serializable. Let us use PWSR to denote the set of histories that are predicatewise serializable. Then, we have $QSR \subseteq PWSR$.

**Theorem 3.5** $QSR \subset PWSR$.

*Proof:* We need to show that $QSR \neq PWSR$. This is true because the global history $H$ in example 3.2 is predicatewise serializable, but not quasi serializable. □

# 4 A Correctness Proof of Altruistic Locking Algorithm

It has been our goal in this paper to present a more flexible correctness criterion than serializability. This allows us to validate algorithms which provide a high degree of concurrency and do not violate local autonomies. In this section, we apply our quasi serializability theory to Alonso's altruistic locking algorithm

[AGS87]. This algorithm was chosen for illustration because it was, we felt, correct, non-serializable and clearly stated.

## 4.1 How the Algorithm Works

The basic idea of the algorithm is to use global locking to coordinate the executions of global transactions at local sites. To improve the performance, the altruistic locking protocol provides a mechanism for global transactions to release locks before they finish.

Specifically, the altruistic locking protocol works as follows. A global transaction must lock a site before it can request work from that site. Once the global transaction's request has been processed, and if the global transaction will request no further work from that site, it can release its lock on the site. Other global transactions waiting to lock the released site may be able to do so if they are able to abide by the following restrictions. The set of sites that have been released by a global transaction constitutes the *wake* of that transaction. A global transaction is said to be in another global transaction's wake if it locks a site which is in that transaction's wake. The simplest altruistic locking protocol says that a global transaction running concurrently with another global transaction must either remain completely inside that transaction's wake, or completely outside its wake, until that transaction has finished.

## 4.2 A Non-Serializable Example

Although the authors of the altruistic locking algorithm believed that the algorithm ensures serializability of the global executions [AGS87], it actually does not. To see this, let us consider the following example.

**Example 4.1** Consider an HDDBS consisting of two local databases, $\text{LDBS}_1$ and $\text{LDBS}_2$, where data items $a$ and $b$ are at $\text{LDBS}_1$, and data items $c$ and $d$ are at $\text{LDBS}_2$. The following global transactions are submitted to the system:

$G_1 : r_{g_1}(a) w_{g_1}(c)$

$G_2 : w_{g_2}(b) r_{g_2}(d)$

Let $L_1$ , $L_2$ be the local transactions at $\text{LDBS}_1$ and $\text{LDBS}_2$, respectively:

$L_1 : w_{l_1}(a)r_{g_2}(b)$

$L_2 : r_{l_2}(c)w_{l_2}(d)$

Let $H_1$ and $H_2$ be the local histories at $\text{LDBS}_1$ and $\text{LDBS}_1$, respectively:

$H_1 : w_{l_1}(a)r_{g_1}(a)w_{g_2}(b)r_{l_1}(b)$

$H_2 : w_{g_1}(c)r_{l_2}(c)w_{l_2}(d)r_{g_2}(d)$

Suppose $G_1$ locks $\text{LDBS}_1$ before $G_2$ does. $G_2$ waits until $G_1$ finishes reading data item $a$ and releases $\text{LDBS}_1$. The same thing happens at $\text{LDBS}_2$. Now suppose that $G_1$ gets the lock first. After updating data item $b$, $G_1$ releases the lock. $G_2$ then gets the lock and reads data item $b$. Since $G_2$ is completely in the wake of $G_1$, the history may be generated (or certified) by the algorithm. It is not hard to see that the global history, $H = \{H_1, H_2\}$, is not serializable. □

## 4.3 Correctness Proof

We prove that altruistic locking is a correct concurrency control algorithm (for HDDBSs defined in section 2) by proving that all global histories generated (or certified) by the algorithm are quasi serializable. We do so by constructing an acyclic QSG for every global history.

Let $H$ be a global history over $G \cup L$, where $G = \{G_1, G_2, ..., G_n\}$ is a set of global transactions ($n > 1$). Suppose that $H$ is generated (or certified) by the altruistic locking algorithm.

Let $G_i$ and $G_j$ be two global transactions in $G$ which access a common database. If $G_j$ gets any lock first, then either $G_i$ is completely in the wake of $G_j$ or $G_i$ will not start until $G_j$ has finished. Otherwise (if $G_i$ gets any lock first), either $G_j$ is completely in the wake of $G_i$ or $G_j$ will not start until $G_i$ has finished.

Since $G_i$ conflicts with $G_j$, one of $G_i$'s operations must conflict (directly or indirectly) with one of $G_j$'s operations. Recall that an operation conflicts with another operation only if it precedes that operation in a history. One of $G_i$'s operations must precede one of $G_j$'s operations in a local history. Therefore, we have:

**Lemma 4.1** *If the edge $G_i \rightarrow G_j$ $(1 < i, j < n)$ is in QSG(H), then all $G_i$'s operations precede $G_j$'s operations in all local histories.*

**Theorem 4.1** *QSG(H) is acyclic.*

*Proof:* Suppose that there is a cycle in QSG($H$): $G_{i_1} \rightarrow G_{i_2} \rightarrow ... \rightarrow G_{i_k} \rightarrow G_{i_1}$, where $1 \leq i_1, i_2, ..., i_k, k \leq n$. By lemma 4.1, all of $G_{i_1}$'s operations precede $G_{i_2}$'s operations at all local sites, all of $G_{i_2}$'s operations precede $G_{i_3}$'s operations at all local sites, etc., and all of $G_{i_k}$'s operations precede $G_{i_1}$'s operations at all local sites. In other words, all of $G_{i_1}$'s operations precede $G_{i_1}$'s operations at all local sites, a contradiction! □

# 5 Conclusion

We have extended serializability to verify the correctness of concurrency control algorithms for HDDBSs, resulting in *quasi serializability*. A global history in an HDDBS is quasi serializable if it is (conflict) equivalent to a *quasi serial* history in which all global transactions are submitted sequentially. We have proved that a global history is quasi serializable if and only if it has an acyclic *quasi serialization graph*. We have used this result to prove the correctness of an altruistic locking algorithm.

The main difference between quasi serializability and general serializability theories is that the latter treats global and local transactions equally while the former treats them differently. More specifically, quasi serializability theory is based primarily (not totally) on the behaviors of global transactions. This makes global concurrency control easier. One immediate observation is that the global concurrency controller can ensure the correctness of the global history (i.e. quasi serializability) by simply controlling the submission of global transactions (e.g., serially).

Quasi serializability is intended for use as a correctness criterion for global concurrency control in InterBase. However, it can also be used in any HDDBSs that meet the assumptions given in section 2. As a matter of fact, it can even

be used in replicated HDDBSs as long as local transactions do not update any replicated data items.

**Acknowledgements**

The authors would like to acknowledge the contributions made by Yongho Leu and Shawn Ostermann. They have contributed immensely to the content and presentation of this paper.

# References

[AGS87] R. Alonso, H. Garcia-Molina, and K. Salem. Concurrency control and recovery for global procedures in federated database systems. In *IEEE Data Engineering*, pages 5–11, September 1987.

[BG85] P. Bernstein and N. Goodman. Serializability theory for database. *Journal of Computer and System Sciences*, 31(3):355–374, 1985.

[BS88] Y. Breitbart and A. Silberschatz. Multibdatabase update issues. In *Proc. Int'l Conf. On Management of Data*, pages 135–142, June 1988.

[BST87] Y. Breitbart, A. Silberschatz, and G. Thompson. An update mechanism for multidatabase systems. *IEEE Data Engineering*, 10(3):12–18, September 1987.

[DELO88] W. Du, A. Elmagarmid, Y. Leu, and S. Ostermann. *Effects of Autonomy on Maintaining Global Serializability in Heterogeneous Database Systems.* Technical Report CSD-TR-835, Department of Computer Science, Purdue University, December 1988.

[EH88] A.K. Elmagarmid and A.A. Helal. Supporting updates in heterogeneous distributed database systems. In *Proceedings of the International Conference on Data Engineering*, pages 564–569, 1988.

[EV87] F. Eliassen and J. Veijalainen. Language support for mutidatabase transactions in a cooperative, autonomous environment. In *TENCON '87, IEEE Regional Conference*, Seoul, 1987.

[GL84] V.D. Gligor and G.L. Luckenbaugh. Interconnecting heterogeneous data base management systems. *IEEE Computer*, 17(1):33–43, January 1984.

[GP85] V.D. Gligor and R. Popescu-Zeletin. Concurrency control issues in distributed heterogeneous database management systems. In F.A. Schreiber and W. Litwin, editors, *Distributed Data Sharing Systems*, North-Holland, 1985.

[GP86] V.D. Gligor and R. Popescu-Zeletin. Transaction management in distributed heterogeneous database management systems. *Inform. Systems*, 11(4):287–297, 1986.

[KS88] H. K. Korth and G. D. Speegle. Formal model of correctness without serializability. In *Proc. Int'l Conf. On Management of Data*, pages 379–386, June 1988.

[LEM88] Y. Leu, A. Elmagarmid, and D. Mannai. *A transaction management facility for InterBase*. Technical Report TR-88-064, Computer Engineering program, Pennsyvania State University, May 1988.

[Pu86] C. Pu. *Superdatabases for Composition of Heterogeneous Databases*. Technical Report CUCS-243-86, Columbia University, 1986.

[RSI78] D. Rosenkrantz, R. Stearns, and P. Lewis II. System level concurrency control for distributed database systems. *ACM Transactions on Database Systems*, 3(2):178–198, June 1978.

[SIR76] R. Stearns, P. Lewis II, and D. Rosenkrantz. Concurrency control for database systems. In *Proceeding of the 17th Annual Symposium on Fundations of Computer Science*, pages 19–32, October 1976.

[SK80] A. Silberschatz and Z. Kedem. Consistency in hierarchical database systems. *JACM*, 27(1):72–80, 1980.

[Sug87] K. Sugihara. Concurrency control based on cycle detection. In *Proc. Int'l Conf. On Data Engineering*, pages 267–274, February 1987.