Quasi Serializability: a Correctness Criterion for Global Concurrency Control in InterBase*

Weimin Du and Ahmed K. Elmagarmid
Computer Sciences Department
Purdue University
West Lafayette, IN 47907

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.

1 Introduction

The InterBase project in the Computer Sciences 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 heterogeneous.

HDDBSs have become a very attractive research area recently. Within this research area, the transaction processing problem, especially that of concurrency control, has received considerable attention [GL84] [GP86] [Pu86] [BS88] [EH88] [LEM88]. Unfortunately, no satisfactory global concurrency control algorithm has been given yet. The lack of suitable theoretical tools for proving the correctness of these algorithms is one of the reasons.

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 autonomy of the element databases. As a result, some of the proposed algorithms violate local autonomy (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], [EH88] and [LEM88]).

The hierarchical nature of global concurrency control and the fact that local databases are autonomous make it very difficult to maintain global serializability in HDDBSs. On the other hand, the global concurrency controller (GCC) is relieved 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

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

ones. 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 an altruistic locking algorithm [AGS87]. The correctness of this algorithm cannot be shown using serializability. Some concluding remarks are given in section 5.

2 Background

2.1 An HDDBS Model

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 LDBS 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 LDBS 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 LDBS D_i) is defined over global subtransactions $G_{1,i}, G_{2,i}, ..., G_{n,i}$, and local transactions $L_{i,1}, ..., L_{i,j_i}$.

Informally, an HDDBS is a collection of autonomous database systems wishing to cooperate. The element databases are connected and coordinated by a global database management system. Concurrency control is performed hierarchically. The local concurrency controller (LCC) at each LDBS is responsible for the correctness of the local history at that site. The GCC, which is added on the top of the existing LDBSs, coordinates the local histories at all local sites to guarantee the correctness of the global history. To simplify GCC's work, a global transaction is allowed to submit at most one subtransaction to each local site [GP86].

2.2 Assumptions

In this subsection, we list and explain the assumptions we have made in the study of QSR. Some of them are general in HDDBSs (e.g., local autonomy), while the rest are less general and are made to simplify the consistency problem of HDDBSs.

Local Autonomy

In an HDDBS, LDBSs are assumed to be autonomous. Autonomy is defined as follows [EV87] [DELO88].

- *Design autonomy:* Each of the LDBSs is free to use whatever data models and transaction management algorithms it wishes.
- *Communication autonomy:* Each of the LDBSs is free to make independent decisions as to what information to provide to the GCC or other LDBSs, and when to provide it.
- *Execution autonomy:* Each of the LDBSs is free to do anything (e.g., commit or restart) on any transactions running at its local site.

Basically, local autonomy reflects the fact that LDBSs were designed and implemented independently, and were totally unaware of the integration process. Local autonomy has significant effects on global concurrency control and the meaning of global consistency [DELO88].

Data Integrity Constraint

Another assumption we have made is that there is no data integrity constraint on data items at different sites. This assumption can be thought of as a consequence of local autonomy. Since each LDBS is designed and implemented independently, there is no relationship between data items at different LDBSs at all. In addition, it is also impossible to preserve these global data integrity constraints because neither the local users nor the LCCs of each LDBS are aware of the integration process and therefore the constraints[1].

As a result, global replication can only be in restrictive forms. In this paper, we assume that only those data items that are not directly updatable by local transactions are allowed to be replicated.

Intra Transaction Dependency

In this paper, we also assume that there is no value dependency between subtransactions of the same global transaction. This assumption is made because of the following consideration. In HDDBSs, it is very expensive (if not impossible) to support value dependency between subtransactions. Recall that LDBSs are

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

usually connected through a global database management system, and usually do not have direct communication. Since a global transaction can only submit one subtransaction to each site, the only way to implement value dependency is to submit those subtransactions with value dependency relations sequentially, even though they are executed at different sites.

We realized that this assumption is not necessary for QSR. However, the description of a less restrictive condition requires more concepts and work on the transaction model, and therefore is out of the scope of this paper (we refer readers to [ED89]). In addition, with this assumption, the appropriateness of QSR can be explained more easily and clearly.

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., [BHG87]).

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 and 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 if 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 *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 [BHG87].

Definition 3.2 *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 LDBSs, D_1 and D_2, where data items a and b are at D_1, and c, d and e are at D_2. The following global transactions are submitted to the HDDBS:

$G_1 : w_{g_1}(a)r_{g_1}(d)$ $\quad$ $G_2 : r_{g_2}(b)r_{g_2}(c)w_{g_2}(e)$

Let L_1 and L_2 be some local transactions submitted at D_1 and D_2, respectively:

$L_1 : r_{l_1}(a)w_{l_1}(b) \qquad L_2 : w_{l_2}(d)r_{l_2}(e)$

Let H_1 and H_2 be local histories at D_1 and D_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*** of a global history H, denoted QSG(H), is a directed graph whose nodes are the global transactions in H, and whose edges are all the relations (G_i, G_j) $(i \neq j)$ such that $G_i \rightarrow^H G_j$.

QSG(H) is completely determined by the conflict relations of H. Therefore, QSG(H)=QSG(H') if H and H' are conflict equivalent. However, the reverse might not be true because QSG(H) loses some information of local transactions. Nevertheless, QSG(H) contains (in indirect conflict relations) enough information about local transactions for the testing of quasi serializability, as the following theorem indicates.

Theorem 3.1 *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. 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_l $(1 \leq l \leq m)$, assume that $G_{i_1,l}, G_{i_2,l}, ..., G_{i_n,l}$ are the global subtransactions that appear in H_l. We show, below, that there is another serializable local history H_l', equivalent to H_l, such that all of $G_{i_1,l}$'s operations precede $G_{i_2,l}$'s operations in H_l', all of $G_{i_2,l}$'s operations precede $G_{i_3,l}$'s operations in H_l', and so on.

In order to construct the equivalent history H_l', let us group the operations in H_l into n operation sets based on global subtransactions.

```
For $p = 1$ to $n - 1$ do
    $OP(i_p, l) = \{ o \in H_l - \bigcup_{k=1}^{p-1} OP(i_k, l)$ :
                  a. $o \in G_{i_p,l}$, or
                  b. $o$ conflicts with one of
                     $G_{i_p,l}$'s operations $\}$
$OP(i_n, l) = \{$ everything left $\}$
```

Informally, $OP(i_p, l)$ consists of all $G_{i_p,l}$'s operations and those operations in H_l that must precede some of $G_{i_p,l}$'s operations in any history which is conflict equivalent to H_l (but not in previous OP's). OPs are well defined and have the following properties:

1. $OP(i_p, l)$ contains all operations of global subtransaction $G_{i_p,l}$, but no operation of other global subtransactions, where $1 \leq p \leq n$.
2. $\bigcup_{k=1}^{n} OP(i_k, l)$ consists of all the operations in H_l.
3. $OP(i_p, l) \cap OP(i_q, l) = \emptyset$, where $1 \leq p \neq q \leq n$.

Let SH be the constructor that constructs from an OP a subhistory which has the same conflict relations as H_l as far as the operations in OP are concerned. This can be done by ordering all operations in OP in the same way as in H_l.

Now H_l' can be constructed as follows:

$$SH(OP(i_1, l)) \circ SH(OP(i_2, l)) \circ ... \circ SH(OP(i_n, l))$$

where $\circ$ stands for concatenation of subhistories. We claim that:

1. H_l' involves the same transactions as H_l;
2. H_l' is conflict equivalent to H_l; and
3. Global subtransactions in H_l are executed sequentially in H_l'.

The correctness of the first and the last statements are clear. We now show that the second statement is also true.

Let o_i and o_j be two operations in H_l and o_i conflicts with o_j. There exists integer p such that $o_j \in OP(i_p, l)$. If $p < n$, then o_j either belongs to $G_{i_p,l}$ or conflicts with one of $G_{i_p,l}$'s operations, and so is o_i. Therefore, either $o_i \in OP(i_q, l)$, where $q < p$, or $o_i \in OP(i_p, l)$ by the definition of $OP(i_p, l)$. In either case, o_i precedes o_j in H_l'. This is also true when $p = n$. So, o_i also conflicts with o_j in H_l'.

Let $H' = \{H_1', H_2', ..., H_m'\}$, then H' is quasi serial and equivalent to H. Therefore, H is quasi serializable.

(only if) Let H be a quasi serializable global history. Again, we assume that $G_1, G_2, ..., G_n$ are the global transactions in H. Let H' be a quasi serial global history which is equivalent to H. Then QSG(H) = QSG(H'). Since $G_1, G_2, ..., G_n$ are executed sequentially in H' and one operation can only conflict with subsequent operations in a history, QSG(H') must be acyclic and so does QSG(H). □

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. □

$$\begin{tikzcd}
& L_1 \arrow[dr] & \\
G_1 \arrow[ur] & & G_2 \arrow[dl] \\
& L_2 \arrow[ul] &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
G_1 \arrow[r] & G_2
\end{tikzcd}$$

(b)

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.

3.3.1 HDDBS Consistency Problem

It is generally accepted (see, e.g., [SLR76]) 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 consistency constraints. There are two types of consistency constraints associated with a database system:

1. Data integrity constraints
   These constraints are defined on data items and 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. Transaction consistency constraints
   These 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.

Data integrity constraints 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 (assume that each transaction, when executing alone, will preserve these constraints.)

Transaction consistency constraints, on the other hand, usually cannot be expressed in predicate form. As with data integrity 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 LDBS has its own set of consistency constraints. When the LDBSs are integrated into an HDDBS, these consistency constraints are combined, together with some additional consistency constraints for the global database, to form the consistency constraints of the HDDBS.

In our HDDBS environment (as described in section 2), the consistency problem is different from what we have described above. The main difference is that there is no additional global data integrity constraint except the mutual consistency constraints for globally replicated data items.

On the other hand, there might be global transaction consistency constraints. This is possible because these constraints can be totally independent of any local transactions (e.g., constraints between two global transactions). We are currently investigating these and related problems. This is also necessary, as the following example shows.

Example 3.3 Consider an HDDBS consisting of two LDBSs, D_1 and D_2, where data item a is at D_1 and data items b and c are at D_2. The following global transactions are submitted to the HDDBS:

$G_1 : w_{g_1}(a)r_{g_1}(c) \qquad G_2 : r_{g_2}(a)w_{g_2}(b)$

Let L be the local transaction submitted at D_2:

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

Let H_1 and H_2 be local histories at D_1 and D_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 summary, an HDDBS is consistent if and only if (1) it satisfies the global transaction consistency constraints (2) the globally replicated data items are consistent, and (3) all the LDBSs are consistent.

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.

First, a quasi serial history preserves the mutual consistency of all globally replicated data items. This is true because they are updated only by global transactions that are executed serially.

Second, a quasi serial history preserves the global transaction consistency constraints. 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 transaction consistency constraints 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 are executed serially.
- Constraints on local transactions (or global subtransactions) at different LDBSs: There is no direct interference between local transactions at different LDBSs because they do not access any common data item. Global transactions might introduce indirect interference between them. However, this is true only if there exist value dependency between subtransactions of the same global transaction, which is not the case in our environment. Therefore, constraints on local transactions at different LDBSs are always preserved because there is no interference between these local transactions.

Example 3.4 Let us consider the global history H in example 3.1. Since there is no value dependency between the two subtransactions of transaction G_2, the value of data item e written by G_2 at D_2 is not related to the value of data item b read by G_2 at D_1. Therefore, the value of data item e read by local transaction L_2 at D_2 is not related to the value of data item b written by local transaction L_1 at D_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 integrity constraints can be defined on data items at different LDBSs. A quasi serializable history might not preserve these constraints. Second, it is possible for subtransactions of the same global transaction to have value dependency in homogeneous environments because they can communicate with each other. Therefore, two local transactions at different LDBSs might interact with each other. Unless all the transactions are executed in a serializable way, undesired interference between local transactions at two LDBSs might happen, and therefore constraints on these local transactions will not be preserved.

3.4 Comparison to Other Criteria

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 [BHG87]. 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

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

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 (conflict) serialization graph. Therefore, a conflict serializable global history is also quasi 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 LDBSs, D_1 and D_2, where data items a, b and c are at D_1 and d is at D_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 D_1.

$L_1 : w_{l_1}(a)w_{l_1}(b) \qquad L_2 : w_{l_2}(a)w_{l_2}(b)$

Let H_1 and H_2 be local histories at D_1 and D_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 [BHG87]), 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.

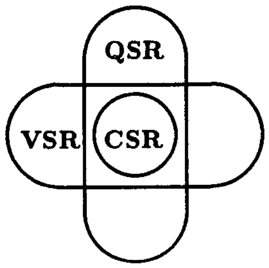

Figure 2: Relationships among QSR, CSR and VSR.

In [KS88], Korth proposed the use of predicatewise serializability (PWSR) as the correctness criterion for concurrency control in CAD database and office information systems. The basic idea of PWSR is that if the 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, PWSR concerns data integrity constraints only. In HDDBS 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 also 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 an 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 an 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 LDBSs, D_1 and D_2, where data items a and b are at D_1, and data items c and d are at D_2. The following global transactions are submitted to the system:

$G_1 : r_{g_1}(a)w_{g_1}(c) \qquad G_2 : w_{g_2}(b)r_{g_2}(d)$

Let L_1 , L_2 be the local transactions at D_1 and D_2, respectively:

$L_1 : w_{l_1}(a)r_{l_1}(b) \qquad L_2 : r_{l_2}(c)w_{l_2}(d)$

Let H_1 and H_2 be the local histories at D_1 and D_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 D_1 before G_2 does. G_2 waits until G_1 finishes reading data item a and releases D_1. The same thing happens at D_2. In other words, G_1 gets the lock first. After updating data item c, G_1 releases the lock. G_2 then gets the lock and reads data item d. 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.

Suppose that G_i conflicts with G_j. Then 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 shown 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 is appropriate in HDDBS environments because of the fact that there is no global data integrity constraint and no value dependency between subtransactions of the same global transaction.

Quasi serializability makes global concurrency control easier in the sense that it allows a higher concurrency than serializability. 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). However, it is not obvious how to take advantage of quasi serializability to design global concurrency control protocols other than submitting global transactions serially. Further work is still needed in this area.

Another advantage of quasi serializable histories is the compatibility of (quasi) serialization and execution orders of the global transactions. This property is very useful for multi-level transaction management protocols where concurrency control is performed hierarchically and independently at different levels [BSW88].

Quasi serializability is intended to be used as a correctness criterion for global concurrency control in InterBase. It can also be used in any database system that meets the assumptions given in section 2. As a matter of fact, it can even be used in more general database systems (e.g., allowing restrictive value dependency between subtransactions of the same global transaction). We are now working on the problem of defining less restrictive transaction models suitable for quasi serializability and will report the result elsewhere.

Acknowledgements

The authors would like to acknowledge the contributions made by Yungho 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 Bulletin*, pages 5–11, September 1987.

[BHG87] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Databases Systems.* Addison-Wesley Publishing Co., 1987.

[BS88] Y. Breitbart and A. Silberschatz. Multibdatabase update issues. In *Proceeding of the International Conference on Management of Data*, pages 135–142, June 1988.

[BSW88] C. Beeri, H. Schek, and G. Weikum. Multilevel transaction management, theoretical art or practical need? In *Proceeding of the International Conference on Extending Database Technology*, pages 134–154, March 1988.

[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, Purdue University, December 1988.

[ED89] A. Elmagarmid and W. Du. *Nested Transactions with Value Dependency Restrictions.* Technical Report CSD-TR-885, Purdue University, May 1989.

[EH88] A. Elmagarmid and 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. Gligor and G. Luckenbaugh. Interconnecting heterogeneous data base management systems. *IEEE Computer*, 17(1):33–43, January 1984.

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

[KS88] H. Korth and G. Speegle. Formal model of correctness without serializability. In *Proceeding of the International Conference 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.

[SLR76] R. Stearns, P. LewisII, and D. Rosenkrantz. Concurrency control for database systems. In *Proceeding of the Annual Symposium on Fundations of Computer Science*, pages 19–32, October 1976.

[Sug87] K. Sugihara. Concurrency control based on cycle detection. In *Proceeding of the International Conference on Data Engineering*, pages 267–274, February 1987.