Pergamon

Information Systems Vol. 22, No. 6, pp. 387–400, 1997

0306-4379/97 $15.00 + 0.00

ON RELAXING SERIALIZABILITY BY CONSTRAINING TRANSACTION READSETS†

EVAGGELIA PITOURA[1], AIDONG ZHANG[2] and BHARAT BHARGAVA[3]

[1]Computer Science Department, University of Ioannina, GR 45110 Ioannina, Greece
[2]Department of Computer Science, SUNY at Buffalo, Buffalo, NY 14260
[3]Department of Computer Sciences, Purdue University, West Lafayette, IN 47907

(Received 20 October 1995; in final revised form 17 September 1997)

Abstract — Although, concurrency control in database systems is primarily based on serializability, many recent applications have rendered traditional serializability-based criteria inefficient or inappropriate. However, non-serializable executions may violate database consistency. In this paper, we propose a new approach to ensuring the correctness of non-serializable executions. The approach is based on relating transaction views of the database to the integrity constraints of the system. The underlying concepts of view closure and view consistency are defined. Then, drawing upon this approach, we develop a new correctness criterion for multidatabases, which are confederations of pre-existing heterogeneous and autonomous distributed database systems. This criterion, called view-based two-level serializability, relaxes serializability while respecting the autonomy of local database systems and preserving multidatabase consistency. We investigate the application of the criterion to various practical multidatabase scenarios and discuss implementation issues. Copyright ©1997 Elsevier Science Ltd

Key words: Multidatabases, Concurrency Control, Transaction Management, Database Consistency, Integrity Constraints

1. INTRODUCTION

Database concurrency control deals with the maintenance of database consistency in the presence of concurrently executing transactions based on the premise that each transaction maintains database consistency when executed in isolation. In most database concurrency control methods, database consistency is ensured by requiring that the resulting interleaved execution of transactions is serializable, that is, equivalent to some serial execution of the transactions. While this approach is attractively simple, many recent applications have rendered it inefficient or inapplicable [5, 17]. For example, in applications where transactions are long-running, such as in computer-aided design, maintaining serializability causes unbearable overheads. Besides performance considerations, relaxing serializability is also motivated by the interactive nature of many complex applications. In cases of applications sharing intermediate results, such as in cooperative environments or in agent-based computing, serializable executions are not only expensive but also undesirable. However, nonserializable executions may lead to inconsistent database states.

In this paper, we propose a new approach for ensuring correctness of nonserializable executions. The proposed approach draws upon the observation that the view of a transaction, that is the data it reads, plays an important role on the maintenance of consistency. The underlying concepts of view consistency and view closure of transactions are defined by relating the transaction views to the integrity constraints of the system. Intuitively, a transaction view is consistent with respect to a set of data items of type D if the data of type D read by the transaction can be part of a consistent database state, and is closed with respect to a set of constraints C if, when the transaction reads a data item a, it also reads all data items whose values depend on a through a type C constraint. This approach is then applied to multidatabase systems.

A multidatabase system, MDBS, is comprised of multiple distributed and possibly heterogeneous database systems that cooperate in an autonomous fashion [20, 15]. Multidatabase systems are getting increasingly important with the advance of internetworking and the desire to build

†Recommended by Zvi M. Kedem

world-wide information servers. In an MDBS, concurrency control is performed at two levels, locally by the pre-existing local transaction managers, LTMs, of the participating database systems, and globally by a global transaction manager, GTM. LTMs are responsible for the correct execution of all transactions executed at their sites. There are two types of transactions, local and global. Local transactions access data at one site only, are submitted to the appropriate LTM, and are executed outside the control of the GTM. Global transactions are submitted to the GTM, where they are parsed into a number of global subtransactions, each of which accesses data stored in a single local database. These subtransactions are then submitted for execution to the appropriate LTM. The GTM retains no control over global subtransactions after their submission to the LTMs. The only assumption about the execution of transactions at each local site is that it is serializable. The autonomous nature of the LTMs greatly complicates the problem of transaction management in a multidatabase system.

Concurrency control in MDBSs has received much attention from multidatabase researchers. In particular, maintaining global serializability in the execution of both local and global transactions has been well studied [3]. As observed in the proposed approaches, global serializability can be retained by ensuring that the serialization orders of global subtransactions at all local sites are consistent with each other. The difficulties in this regard center upon the inability of the GTM to control the serialization order of global subtransactions at local sites due to possible indirect conflicts with local transactions. In general, all successful attempts for ensuring global serializability require the enforcement of conflicting operations among global subtransactions at each local site [7, 12, 21]. The GTM can thus control the serialization order of global subtransactions by controlling the execution of these conflicting operations. However, enforcing conflicts may result in poor performance if most global transactions would not naturally conflict. Relaxing global serializability is thus a significant issue for multidatabase concurrency control [4, 16, 14].

A practical non-serializable criterion, called two-level serializability (2LSR), was introduced in [14]. 2LSR, in addition to the assumption of serializability of transactions at each local site provided by the LTMs, requires the serialization at the GTM level of the global transactions only. Thus, 2LSR can be easily enforced without violating local autonomy. However, 2LSR executions of local and global transactions do not, in general, guarantee database consistency. In this paper we use the view-based approach to ensure the correctness of 2LSR executions. Specifically, we state and prove exact closures and consistency conditions that must be imposed on transaction views so that 2LSR schedules preserve database consistency. These conditions are then refined for two specific multidatabase scenarios.

The remainder of this paper is organized as follows. Section 2 describes the multidatabase model and motivates our approach. Section 3 introduces the concept of view closure and consistency. In Section 4, we present the formulation of view-based correctness of 2LSR schedules, and show how some of the proposed conditions may be relaxed for practical multidatabase models. Section 5 discusses some important characteristics of the approach and implementation issues. Section 6 compares the present work with related nonserializable criteria in terms of the range of acceptable schedules and of its applicability to a multidatabase environment. Finally, concluding remarks are offered in Section 7.

2. RELAXING SERIALIZABILITY

In this section, we first introduce the multidatabase model and then discuss the impact of relaxing global serializability on multidatabase consistency.

2.1. Multidatabase Consistency

A multidatabase is the union of all data items stored at the participating local sites. We denote the set of all data items in a local site LS_i by D_i for $i = 1, ..., m$ and the set of all data items in the multidatabase by $\mathcal{D}$. Thus, $\mathcal{D} = \bigcup_{i=1}^{m} D_i$. We assume that local databases are disjoint; that is, $D_i \cap D_j = \emptyset, i \neq j$. Following the traditional approach, a *database state* is defined as a mapping of every data item to a value of its domain, and integrity constraints are formulas in predicate calculus that express relationships of data items that a database must satisfy. The consistency of a database state is then defined as follows:

Definition 1 A database state is *consistent* if it preserves all integrity constraints defined in the MDBS environment.

A *transaction* is a sequence of *read* and *write* operations resulting from the execution of a transaction program. We denote the read and write operations as $r(x)$ and $w(x)$ (possibly subscripted) or alternatively use $r(x,v)$ (or $w(x,v)$) to denote an operation which reads (or writes) a value v from (or to) the data item x. Two operations *conflict* with each other if they access the same data item and at least one of them is a *write* operation. The execution of a transaction transfers a database from one consistent state to another.

The set of data items at a local site is partitioned into *local* data items, denoted LD_i that correspond to data items prior to multidatabase integration and *global* data items, denoted GD_i, that correspond to data created after the integration such that $LD_i \cap GD_i = \emptyset$ and $D_i = LD_i \cup GD_i$. The set of all global data items is denoted GD, $GD = \bigcup_{i=1}^{m} GD_i$. We distinguish two types of transactions local and global transactions. *Local transactions* access data items at a single local site and are outside the control of the GTM. Local transactions model local applications that existed prior to the integration and thus access only local data at the corresponding component database. Local transactions correspond also to programs written after the integration that are scheduled to be executed without any global control for reasons of efficiency, privacy, or autonomy. In this case, local transactions may also read global data, however, since they are only under local control, they are unable to maintain integrity constraints that span more than one site, and thus are not allowed to update global data. A *global transaction* is submitted to the GTM and reads and writes both local and global data at multiple sites. Each global transaction is decomposed by the GTM into a set of *global subtransactions*, each of which accesses data stored in a single local database. After the submission of a global subtransaction at the local site, the GTM has no control over it.

We assume that there are not any integrity constraints that involve remote (i.e., located at different sites) local data, or both remote local and global data, since then pre-existing local transactions being unaware of them would provide no guarantees for their preservation. To support these types of constraints special techniques have been proposed that are orthogonal to this paper [19, 9]. Thus, three types of integrity constraints are possible: *local integrity constraints* defined on local data items at a single local site; *local/global integrity constraints* defined between local and global data items at a single site; and *global integrity constraints* defined on global data items that may be located at different sites.

A *schedule* over a set of transactions is a partial order of all and only the operations of those transactions which orders all conflicting operations and which respects the order of operations specified by the transactions. In a MDBS environment, a *local schedule* S^{D_k} is a schedule over both local transactions and global subtransactions which are executed at the local site LS_k, and a *global schedule* S is a schedule over both local and global transactions which are executed in an MDBS. A *global subschedule* $S^{\mathcal{G}}$ is global schedule S restricted to the set $\mathcal{G}$ of global transactions in S. The standard assumption in multidatabase concurrency control is that each LTM ensures the serializability of the corresponding local schedule. The correctness of a schedule in terms of database consistency is defined as follows (cf. strong correctness [14] and semantic correctness [6]):

Definition 2 A schedule is *correct* if it preserves all integrity constraints that are defined in the database system and each transaction in S reads a consistent database state.

A global schedule S is considered to be *globally serializable* if S is serializable [2] on the execution of both local and global transactions. Clearly, if local and global transactions maintain all integrity constraints defined in the MDBS environment, then a globally serializable global schedule is correct.

2.2. Motivation

To avoid the potential of poor performance which may be caused by serializability, several researchers have suggested methods of relaxing it. In the case of multidatabases, many researchers

[4, 14] have proposed that the GTM, instead of enforcing global serializability, enforces serializability of the global subschedule, that is of only the global transactions. Enforcing serializability of the global subschedule is straightforward since global transactions, in contrast to local transactions, are under the control of the GTM. Specifically, in [14], the following intuitive non-serializable criterion, termed *two-level serializability* is proposed:

Definition 3 A global schedule is *two-level serializable*, denoted 2LSR, if its global subschedule and local schedules are serializable.

However, as shown by the following example, 2LSR schedules may be incorrect. Note, that inconsistencies may result even when there are no local transactions that read global data and no local/global integrity constraints.

Example 1 Consider an MDBS consisting of two local database systems, where data items a, b, c are at LS_1, and d is at LS_2. Let a, b, c, and d be local data items and the integrity constraints be $a > 0 \rightarrow b > 0$, $c > 0 \rightarrow b > 0$ and $d > 0$. The following two global transaction programs p_1, p_2 and one local transaction program p_L are submitted:

$p_1: \; c = 1$
$\quad\quad if \; d > 0 \; then \; b = 1$
$p_2: \; if \; a > 0 \; then \; d = b \; else \; d = 1$
$p_L: \; a = 1$
$\quad\quad if \; c \leq 0 \; then \; b = 1$

Starting from a consistent database state $a = b = c = -1$, $d = 1$, consider the following execution:

$S_1 : w_L(a, 1) r_2(a, 1) r_2(b, -1) w_1(c, 1) r_l(c, 1)$,
$S_2 : w_2(d, -1) r_1(d, -1)$.

Although the schedule is 2LSR, the resulting database state $a = 1$, $b = -1$, $c = -1$, $d = -1$ is inconsistent. □

The question we pose is whether we can enforce any conditions on the data read by global transaction so that consistency is maintained. Studying the above example, we see that the global transaction resulting from the execution of p_2 reads inconsistent data ($a = 1$, and $b = -1$). Thus, we may conclude that enforcing consistency of the readset of global transactions would suffice. In fact, as proven in Section 4, this is the case when there are no local transactions that read global data. However, this condition proves insufficient, in the presence of such local transactions. The following example is illustrative.

Example 2 Consider an MDBS consisting of two local database systems, where data items a, b, c are at LS_1, and d is at LS_2. Let c be a local data item, a, b, d be global data items, and the integrity constraints be $a > 0 \rightarrow b > 0$ and $d > 0 \rightarrow b > 0$ and $c > 0$. The following two global transaction programs p_1, p_2 and one local transaction program p_L are submitted:

$p_1: \; d = 1$
$\quad\quad b = 1$
$p_2: \; a = 1$
$\quad\quad if \;\; d \leq 0 \; then \; b = 1$
$p_L: \; if \; a > 0 \; then \; c = b \; else \; c = 1$

Starting from a consistent database state $a = b = d = -1$, $c = 1$, consider the following 2LSR execution:

$S_1 : w_2(a, 1) r_L(a, 1) r_L(b, -1) w_1(b, 1) w_L(c, -1)$,
$S_2 : w_1(d, 1) r_2(d, 1)$.

Again, an inconsistent database state $a = 1$, $b = 1$, $c = -1$, $d = 1$ is produced. □

In the above example, both global transactions read consistent data. It turns out, that besides consistency of the readset, appropriate closure conditions are also necessary, since transactions may make false assumptions about values of data they do not read (as p_2 above does for the value of b). In the following sections, we will formally define and prove exact conditions that must be placed on the views of global transactions to ensure that two-level serializable global schedules will preserve database consistency. These conditions will be only placed on global transactions, thus respecting the autonomy of local transactions and sites.

3. TRANSACTION VIEWS

Let t be a transaction. The read set of t, denoted $RS(t)$, is the combination of the set of local data items read by operations in t (denoted $RL(t)$) and the set of global data items read by operations in t (denoted $RG(t)$).

3.1. View Closure

Let c_i be an integrity constraint that is defined on a set of data items $\{d_1, ..., d_l\}$. Let $D(c_i)$ denote the set of data items in c_i. Thus, we have $D(c_i) = \{d_1, ..., d_l\}$. Let $In(d)$ denote the set of data items which share a common integrity constraint with data item d. Clearly, if c_i is the only integrity constraint that is defined in the database, then $In(d_1) = \{d_2, ..., d_l\}$, $In(d_2) = \{d_1, d_3, ..., d_l\}$, and $In(d_i) = \{d_1, ...d_{i-1}, d_{i+1}, ..., d_l\}$ for all $i = 3, ..., l$. We now introduce the concepts of the closure of data items and of transactions which are view-closed on a given set of data items.

Definition 4 Let $D = \{d_1, ..., d_l\}$ be a set of data items. The *closure of* D, denoted $cl(D)$, is the smallest set such that the following conditions are satisfied:

- $D \subseteq cl(D)$.
- If $d \in cl(D)$, then $In(d) \subseteq cl(D)$.

Definition 5 A transaction t is *view-closed* with respect to a set of data items D that it reads if, for any $d \in cl(D)$, $d \in RS(t)$.

A transaction t is *global view-closed* if it is view-closed with respect to the global data items that it reads. A transaction t is *site view-closed* if, at each local site LS_k $(1 \leq k \leq m)$, it is view-closed with respect to the data items that it reads in D_k; that is, for any data $d \in D_k$ that it reads and all data $d' \in cl(\{d\})$ such that $d' \in D_k$, $d' \in RS(t)$.

The size of closure sets can be reduced dramatically if a more elaborate definition of closure is provided. The definition should be such that the closure $cl(a)$ of a data item a includes only those data items whose values are restricted by a. Take, for instance, the constraints $a < b$ and $c < b$. By Definition 4, $b \in cl(a)$, $c \in In(b)$ thus $c \in cl(a)$, although given the value of a we can make no assumptions about the value of c.

A possible way to refine Definition 4 is by taking into consideration the type of constraints when computing closures $cl(D)$, instead of invariantly including in $cl(D)$ all items in $In(d)$ for each $d \in cl(D)$. To reduce the size of closure sets, we can also utilize our knowledge of the value of the data item a. For instance, take a constraint between data items a and b of the form $a > 0 \rightarrow b > 0$. If the value of a read by a transaction t is negative, we do not have to include b in t's view closure. Thus, some data in the view set are only conditionally read based on the values of items already read. Finally, many techniques proposed in the context of optimizing the processes of constraint evaluation and validation [8] can be applied directly to optimize the size of closures. *Constraint subsumption* [10] is such a case. Let $\{C_1, C_2 \ldots, C_n\}$ be a set of constraints. An additional constraint C is *subsumed* by $C_1, C_2 \ldots, C_n$ if, whenever C is violated then so is at least one $C_i \in \{C_1, C_2 \ldots, C_n\}$. Thus, given that $C_1, C_2 \ldots, C_n$ hold, we do not need to check the validity of C. Note, that constraint subsumption is independent of data and data modifications. Applying this idea to the definition of closures, $D(c)$ is not included in the the closure $cl(a)$ of a data item a when c is subsumed by constraints already in $cl(a)$.

3.2. View Consistency

Let DS be the database state of $\mathcal{D}$. The restriction of DS to data items in $D \subseteq \mathcal{D}$ is denoted by DS^D. DS^D is *consistent* if there exists a consistent database state DS_1, such that $DS_1^D = DS^D$. Note, that for the consistency of a restriction DS^D of a database state, it does not suffice to require that all integrity constraints that can be evaluated by the data in D evaluate to true. Take for instance, $\mathcal{D} = \{a, b, c\}$, the subset $D = \{a, c\}$ and the constraints $a > 0 \rightarrow b > 0$ and $b > 0 \rightarrow c < 0$.

The restriction DS^D, $a = 1, c = 1$, of a database state is not consistent although given data in D no constraint can be evaluated. As another example, take the constraints $a = 1 \rightarrow b = 1$ and $c = 2 \rightarrow b = 2$, and the restriction $a = 1$, $c = 2$ of a database state to $D = \{a, c\}$.

Let $read(t)$ denote the database state seen as a result of the read operations in t and $read(t^D)$ denote the database state of D seen as a result of the read operations in transaction t.

Definition 6 The view of a transaction t is *consistent* if $read(t)$ is consistent.

In the MDBS environment, the consistency of various views of transactions is defined as follows: A transaction t is *local view consistent* at LS_i if $read(t^{LD_i})$ is consistent. A transaction t is *global view consistent* if $read(t^{GD})$ is consistent.

4. VIEW-BASED CORRECTNESS OF 2LSR SCHEDULES

In this section, we impose conditions on the view of global transactions so that 2LSR schedules maintain database consistency. The resulting schedules are called *view-based two-level serializable* global schedules. We will first define conditions for the general case that provides more autonomy, that is for the case where autonomous local transactions are allowed to read global data and also read and write local data related through integrity constraints with global data at the same site. Then, we will show how these conditions can be relaxed for other practical database models.

4.1. Background

To develop our criteria we will use the theory advanced in [18]. For completeness we include here the basic lemmas that we subsequently use. Let $\{DS_1\}t\{DS_2\}$ denote that, when transaction t executes from a database state DS_1, it results in a database state DS_2. Without loss of generality, whenever we say $\{DS_1\}t\{DS_2\}$, we assume that it is possible for t to be executed from DS_1. The conditions required to ensure that the execution of a transaction preserves the consistency of the state of a set of data items are specified as follows [14]:

Lemma 1 *Let t be a transaction and $D \subseteq \mathcal{D}$. Let $\{DS_1\}t\{DS_2\}$ and DS_1 be the database state in which t can be executed. If $DS_1^D \cup read(t)$ is consistent, then DS_2^D is consistent.*

We may now relate the consistency of a database state to the execution of transactions. The state associated with a transaction in a schedule is a possible state of the data items that the transaction may have seen. Let $\tau_w(D, S)$ denote the set of transactions in a schedule S that have at least one write operation on some data item in $D \subseteq \mathcal{D}$. Let S be a schedule and $D \subseteq \mathcal{D}$ such that $(S^\tau)^D$ is serializable, where $\tau_w(D, S) \subseteq \tau$. Let $t_1, ..., t_n$ be a serialization order of transactions in $(S^\tau)^D$ and DS_1 be a database state from which S starts. The state of the database before the execution of each transaction, with respect to data items in D, is defined as follows:

$$state(t_i, D, S, DS_1) = \begin{cases} DS_1^D, & \text{if } i = 1 \\ state(t_{i-1}, D, S, DS_1)^{D - WS(t_{i-1}^D)} \cup write(t_{i-1}^D), & \text{if } i > 1 \end{cases}$$

Note that $read(t_i^D) \subseteq state(t_i, D, S, DS)$. Lemma 1 is used to develop the conditions under which each transaction in a schedule reads a database state that is consistent with respect to a set of data items [13]:

Lemma 2 *Let $D \subseteq \mathcal{D}$, S be a schedule, and $\{DS_1\}S\{DS_2\}$. If,*

- *$(S^\tau)^D$ is serializable with serialization order $t_1, ..., t_n$, where $\tau_w(D, S) \subseteq \tau$,*
- *if $state(t, D, S, DS_1)$ is consistent, then $read(t) \cup state(t, D, S, DS_1)$ is consistent for all $t, t \in \tau_w(D, S)$, and*
- *DS_1^D is consistent,*

then $state(t_i, D, S, DS_1)$ is consistent for all $t_i, i = 1, ..., n$.

The above lemma can be modified for the special case where the data items over which integrity constraints are defined are disjoint. To do so we use the following lemma that relates the consistency of the database state to the consistency of its disjoint subsets [18]:

Lemma 3 *Let $C_1, ..., C_l$ be the conjunction ($\wedge$) of integrity constraints, where C_i is defined over the set of data items in $Q_i \subseteq \mathcal{D}$ for all $i = 1, ..., l$ and $Q_i \cap Q_j = \emptyset$ for all $i \neq j$. Let $Q'_i \subseteq Q_i$ and DS be a database state of $\mathcal{D}$. $DS^{Q'_i}$, for all $i, i = 1, ..., l$, is consistent if and only if $\bigcup_{i=1}^{l} DS^{Q'_i}$ is consistent.*

Using the above lemma, we get from Lemma 2 the following corollary [13]:

Corollary 1 *Let $C_1, ..., C_l$ be the conjunction ($\wedge$) of integrity constraints, where C_k is defined over the set of data items in $Q_k \subseteq \mathcal{D}$ for all $k = 1, ..., l$ and $Q_i \cap Q_j = \emptyset$ for all $i \neq j$. Let S be a schedule and $\{DS_1\}S\{DS_2\}$. For any $k = 1, ..., l$, if*

- *$(S^{\tau})^{Q_k}$ is serializable with serialization order $t_1, ..., t_n$, where $\tau_w(Q_k, S) \subseteq \tau$,*
- *$read(t_i^{\mathcal{D}-Q_k})$ is consistent for all $t, t \in \tau_w(Q_k, S)$, and*
- *$DS_1^{Q_k}$ is consistent,*

then $state(t_i, Q_k, S, DS_1)$ is consistent for all $t_i, i = 1, ..., n$.

4.2. Restricting Transaction Views in the General Case

A transaction is called an *update* transaction if it has at least one write operation. The following lemma shows that for global transactions to maintain consistency of global data, they must be site view-closed and view consistent.

Lemma 4 *Let S be a 2LSR schedule. Let $\{DS_1\}S\{DS_2\}$ and $\{DS_1\}$ be consistent. Let $t_1, ..., t_n$ be the serialization order of the global transactions in $(S^{\mathcal{G}})^{GD}$. If all update global transactions are site view-closed and view consistent then $state(t_i, GD, S, DS_1)$ is consistent for all $t_i, i = 1, ..., n$.*

Proof. Since only global transactions write global data, from Lemma 2, it suffices to prove that $read(t) \cup state(t, GD, S, DS_1)$ is consistent for all update global transactions when $state(t, GD, S, DS_1)$ is consistent. We must show that there is a consistent database state DS_0 such that $DS_0^{RS(t)} = read(t)$ and $DS_0^{GD} = state(t, GD, S, DS_1)$. Since $read(t)$ is consistent, there is a consistent database state DS_m such that $DS_m^{RS(t)} = DS^{RS(t)}$, and since $state(t, GD, S, DS_1)$ is consistent, there is a consistent database state DS_l such that $DS_l^{GD} = state(t, GD, S, DS_1)$. Define $DS_0^{RS(t)} = DS_m^{RS(t)}$ and $DS_0^{\mathcal{D}-RS(t)} = DS_l^{\mathcal{D}-RS(t)}$. For the purposes of contradiction, assume that DS_0 is not consistent, then there must be an integrity constraint between $RS(t)$ and $\mathcal{D} - RS(t)$, say involving data items $d_1 \in RS(t)$ and $d_2 \notin RS(t)$ which does not hold. We have the following two cases. Case (1): both d_1 and d_2 belong to GD, then d_1 and d_2 must be consistent, which is a contradiction. Case (2): at least one of them is a local data item, then, since there are no remote constraints involving local data, both d_1 and d_2 must belong to the same local site. Thus, since $RS(t)$ is site view-closed, $d_2 \in RS(t)$, which is a contradiction. □

The following lemma investigates the conditions that must be imposed on the views of global transaction so that local and global transactions maintain the integrity constraints at each database site.

Lemma 5 *Let S be a 2LSR schedule. For a site k, let S^{D_k} be serializable with serialization order $t_1, \ldots, t_n$. Let $\{DS_1\}S\{DS_2\}$ and $\{DS_1\}$ be consistent. If all update global transactions are global view-closed and view consistent, then $state(t_i, D_k, S, DS_1)$ is consistent for all $t_i, i = 1, ..., n$.*

Proof. The proof proceeds as in the previous lemma. From Lemma 2, we must show that $read(t) \cup state(t, D_k, S, DS_1)$ is consistent for all update global and local transactions, when $state(t, D_k, S, DS_1)$ is consistent. If t is a local transaction, this holds, since $read(t) \subseteq state(t, D_k, S, DS_1)$. We will now prove the formula when t is a global transaction.

As above, we define $DS_m^{RS(t)} = DS^{RS(t)}$, $DS_l^{D_k} = state(t, D_k, S, DS_1)$, and prove that there is a consistent database state DS_0 such that $DS_0^{RS(t)} = read(t)$ and $DS_0^{D_k} = state(t, D_k, S, DS_1)$. Define $DS_0^{D_k} = DS_l^{D_k}$ and $DS_0^{\mathcal{D}-D_k} = DS_m^{\mathcal{D}-D_k}$. For the purposes of contradiction, let assume that DS_0 is not consistent, then there must be an integrity constraint involving $d_1 \in D_k$ and $d_2 \notin D_k$. Since, there are no remote constraints involving local data both d_1 and d_2 must be global data. We have the following three cases. Case (1): both d_1 and $d_2 \in RS(t)$ then they must be consistent, which is a contradiction. Case(2): both d_1 and $d_2 \notin RS(t)$, then assume, for the purposes of contradiction, that the values for d_1, d_2 in DS_m violate those in DS_l; d_1 and d_2 can take any value in DS_m, (and thus those in DS_l) unless one of them, say $d_1 \in cl(\{d\})$, for some d in $RS(t)$. Then, d must be a global data, otherwise we would have a remote constraint between d_2 and d. Since $RS(t)$ is global view-closed then $d_1 \in RS(t)$, which is a contradiction. Case (3): exactly one of d_1, d_2 belongs to $RS(t)$, then since $RS(t)$ is global view-closed the other one must also belong to $RS(t)$, which is a contradiction. □

The following corollary illustrates the conditions that must be imposed on the views of global constraints so that local transactions read consistent data.

Corollary 2 *Let S be a 2LSR schedule. Let S^{D_k} be serializable with serialization order $t_1, \dots, t_n$. Let $\{DS_1\}S\{DS_2\}$ and $\{DS_1\}$ be consistent. If all update global transactions are global view-closed and view consistent, then all local transactions read consistent data.*

Proof. Directly from Lemma 5, since $read(t) \subseteq state(t, D_k, S, DS_1)$. □

Now we are ready to prove the conditions under which a 2LSR schedule is correct.

Theorem 3 *Let S be a 2LSR schedule. If all update global transactions are site and global view-closed, and view-consistent then S is correct.*

Proof. Let DS_1 be a consistent database state and $\{DS_1\}S\{DS_2\}$. We need to show that all transactions in S read consistent data and that DS_2 is consistent. By the assumption, all global transactions read consistent data and by Corollary 2 all local transactions read consistent data. Thus, for all transaction t_i in S, $read(t_i)$ is consistent. Now, let S^{D_k} be serializable with serialization order $t_1, ..., t_n$. Since, from Lemma 5, $state(t_n, D_k, S, DS)$ is consistent, there exists a consistent database state DS_3 such that $DS_3^{D_k} = state(t_n, D_k, S, DS)$ and $DS_3^{RS(t_n)} = read(t_n)$. Thus, t_n can be executed in DS_3. Let $\{DS_3\}t_n\{DS_4\}$. Since $DS_3^{D_k} \cup read(t_n)$ is consistent, by Lemma 1, $DS_4^{D_k}$ is consistent. Since $DS_2^{D_k} = DS_4^{D_k}$, $DS_2^{D_k}$ is consistent. Hence, for all $k, k = 1, ..., m$, $DS_2^{D_k}$ is consistent. Similarly, DS_2^{GD} is consistent from Lemma 4. For the purposes of contradiction, assume that DS_2 is not consistent. Then there must be an integrity constraint involving two data items d_1 and d_2 that is violated. If both d_1 and d_2 belong to the same site, say in LS_k, then they belong to D_k and thus are consistent. Otherwise, since there are no remote constraints involving local data, both d_1 and d_2 must belong to GD and thus be consistent. Hence, S is correct. □

4.3. Restricting Transaction Views in the Absence of Local/Global Constraints

In this section, we restrict our model such that no local/global integrity constraints exist. We will show that in this case we can drop the site-closure condition of the views of global transactions and also reduce the view consistency condition to local view consistency.

Since no integrity constraints are present between local and global data items, the integrity constraints can be viewed as $C_1, ..., C_{m+1}$. Here, C_i for $i = 1, ..., m$ are the conjuncts ($\wedge$) of integrity constraints that are defined over the sets of data items in LD_i for $i = 1, ...m$, respectively, and C_{m+1} is the conjunct of integrity constraints that are defined over the set of data items in GD. We now apply Corollary 1 and show that, when global transactions are local view consistent, global transactions read consistent data.

Lemma 6 *Let S be a 2LSR schedule with no integrity constraints present between local and global data items. Let DS_1 be a consistent database state from which S starts. If all update global transactions in S are local view consistent, then,*
(a) for all global transactions t_i in S, $read(t_i)$ is consistent (e.g., local view consistency implies view consistency)
(b) if, in addition all update global transactions in S are global view-closed, then $state(t_i, D_k, S, DS_1)$ is consistent for all $t_i, i = 1, ..., n$.

Proof. (a) Let $t_1, ..., t_n$ be the serialization order of the global transactions in $(S^{\mathcal{G}})^{GD}$. Since the global transactions are local view consistent, $read(t_i^{\mathcal{D}-GD})$ is consistent for all $i = 1, ..., n$. By Corollary 1, $state(t_i, GD, S, DS_1)$ is consistent for all $i = 1, ..., n$. Since $read(t_i^{GD}) \subseteq state(t_i, GD, S, DS_1)$, $read(t_i^{GD})$ is consistent. Thus, by Lemma 3, $read(t_i)$ is consistent for all $i = 1, ..., n$. Then (b) holds from (a) and Lemma 5. □

The following theorem based upon Corollary 2 and Lemma 6 illustrates the conditions under which 2LSR schedules preserve database consistency when there are no integrity constraints between local and global data items.

Theorem 4 *Let S be a 2LSR schedule with no integrity constraints present between local and global data items. If all update global transactions in S are local view consistent and global view-closed, then S is correct.*

Proof. The proof is similar to that of Theorem 3. Let DS_1 be a consistent database state and $\{DS_1\}S\{DS_2\}$. We need to show that all transactions in S read consistent data and that DS_2 is consistent. By Lemma 6, all global transactions read consistent data. Following Lemma 6(b) and Corollary 2, all local transactions also read consistent data. Thus, for all transaction t_i in S, $read(t_i)$ is consistent. Now, let S^{D_k} be serializable with serialization order $t_1, ..., t_n$. Since, from Lemma 6(b), $state(t_n, D_k, S, DS)$ is consistent, $state(t_n, LD_k, S, DS)$ is then consistent. Hence, there exists a consistent database state DS_3 such that $DS_3^{LD_k} = state(t_n, LD_k, S, DS)$ and $DS_3^{RS(t_n)} = read(t_n)$. Thus, t_n can be executed in DS_3. Let $\{DS_3\}t_n\{DS_4\}$. Since $DS_3^{LD_k} \cup read(t_n)$ is consistent, by Lemma 1, $DS_4^{LD_k}$ is consistent. Since $DS_2^{LD_k} = DS_4^{LD_k}$, $DS_2^{LD_k}$ is consistent. Hence, for all $i, i = 1, ..., m$, $DS_2^{LD_i}$ is consistent. Similarly, DS_2^{GD} is consistent. By Lemma 3, DS_2 is consistent. Hence, S is correct. □

To illustrate the above theorem, let's consider Example 2, where there are no local/global integrity constraints. Since $r_1(d)$ in global transaction t_1 in the given global schedule is not global view closed, Theorem 4 cannot be applied. Suppose that we now require the view of t_1 to be closed as $r_1(d)r_1(b)r_1(a)$ and t_1 to be serialized after t_2 in $S_{\mathcal{G}}$. In this example, t_1 and t_2 do not read and write local data, and each global transaction would therefore transfer global data items from one consistent state to another. Hence, the local transaction t_L would read consistent global data and result in a consistent local database state.

4.3.1. Special Case: Local Transactions do not Read Global Data

We now consider the case where local transactions are not allowed to read global data. Thus, any transaction that wants to access global data is treated as a global transaction and is under the control of the GTM. We prove that in this case, we do not need to impose any closure conditions upon the view set of global transactions.

It follows from Corollary 1 and Lemma 6 that, given that global transactions are local view consistent, local transactions read consistent data:

Lemma 7 *Let S be a 2LSR schedule where local transactions do not read global data and there are no integrity constraints present between local and global data items. Let DS_1 be a consistent database state from which S starts. If all update global transactions in S are local view consistent, then, for all local transactions t_i in S, $read(t_i)$ is consistent.*

Proof. Since S^{D_k} is serializable, S^{LD_k} is serializable for all $k = 1, ..., m$. Let $t_1, ..., t_n$ be the serialization order of the transactions in S^{LD_k}. By Lemma 6, $read(t_i^{\mathcal{D}-LD_k})$ is consistent for all $i = 1, ..., n$. By Corollary 1, $state(t_i, LD_k, S, DS_1)$ is consistent for all $i = 1, ..., n$. For any local transaction t_i in S^{LD_k}, since $read(t_i) \subseteq state(t_i, LD_k, S, DS_1)$, $read(t_i)$ is consistent. □

We now are able to demonstrate that, if global transactions are local view consistent, then 2LSR global schedules preserve database consistency.

Theorem 5 *Let S be a 2LSR schedule where local transactions do not read global data and there are no integrity constraints present between local and global data items. If all update global transactions in S are local view consistent, then S is correct.*

Proof. The proof is similar to that of Theorem 3. Let DS_1 be a consistent database state and $\{DS_1\}S\{DS_2\}$. We need to show that all transactions in S read consistent data and that DS_2 is consistent. By Lemma 6(a) and Lemma 7, for all transactions t_i in S, $read(t_i)$ is consistent. Now, let S^{LD_k} be serializable with serialization order $t_1, ..., t_n$. From Corollary 1, $state(t_n, LD_k, S, DS)$ is consistent. Hence, there exists a consistent database state DS_3 such that $DS_3^{LD_k} = state(t_n, LD_k, S, DS)$ and $DS_3^{RS(t_n)} = read(t_n)$. Thus, t_n can be executed in DS_3. Let $\{DS_3\}t_n\{DS_4\}$. Since $DS_3^{LD_k} \cup read(t_n)$ is consistent, by Lemma 1, $DS_4^{LD_k}$ is consistent. Since $DS_2^{LD_k} = DS_4^{LD_k}$, $DS_2^{LD_k}$ is consistent. Hence, for all $i, i = 1, ..., m$, $DS_2^{LD_i}$ is consistent. Similarly, DS_2^{GD} is consistent. By Lemma 3, DS_2 is consistent. Hence, S is correct. □

To illustrate the above theorem we will use Example 1, where all data items are local and no integrity constraints exist between different local sites. However, since both global transactions in the given global schedule have inconsistent local views, Theorem 5 cannot be applied. If we require that $r_1(a)r_1(b)$ and $r_2(d)$ be consistent, then both $w_1(d)$ and $w_2(c)$ would be consistent. As a result, the local transaction would not read inconsistent data, thus resulting in a consistent local database state.

5. DISCUSSION

In this paper, we have advanced a number of criteria to ensure that 2LSR schedules maintain database integrity constraints. In addition, our approach offers an interesting theoretical result that relates consistency and closure properties of transactions to database consistency. The conditions advanced define precisely the size of the readset of a global transaction that suffices to ensure the consistency of the multidatabase. We have developed a range of conditions depending on the type of data and constraints. The more general local transactions are, the more strict the conditions we have to enforce on the view set of global transactions. At one extreme, if local transactions (e.g., transactions outside the control of the GTM) are allowed to access local data related with global data through integrity constraints and also read global data, then we must enforce both consistency and site and global closure of the view set of update global transactions. At the other extreme, if local transactions are allowed to access only local data that have no relation with any global data, then local consistency of the view set of update global transactions suffices.

5.1. Applicability

We have advanced two types of conditions on the view set of update global transactions, *closure* and *consistency* conditions. Since both conditions are imposed only upon global transactions, the autonomy of local sites is being respected. Furthermore, as the concepts of view consistency and view closure rest solely upon the structural properties of the integrity constraints rather than their semantics, such restrictions can be enforced systematically. Closure conditions can be enforced by appending to the beginning of the global transaction read operations on data items which are included in the closure but not read by the global transaction.

The detection of inconsistency is a classical problem to which much attention has been directed [1, 11]. When an update u is executed, it may cause a change of database state ST to ST_u. By applying tests derived from the constraints, the enforcement algorithm verifies that all relevant constraints hold in state ST_u. Note that in the proposed approach, only a subset, and not the whole database, is tested for consistency. Testing consistency is necessary to ensure that global transactions produce consistent data. However, this test can be reduced to testing appropriately defined weaker notions of consistency by taking advantage of the semantics of global transactions. In particular, this test can be relaxed for those global transactions that still produce consistent data even if they do not read exactly consistent data. This notion is similar to the notion of *sensitive transactions* defined by Garcia-Molina [6] as transactions whose output must be based on consistent data and as transactions whose output is seen by users. We do not have to check for exact view consistency of non-sensitive transactions. Our approach can be used in conjunction with methods with relaxed consistency requirements such as epsilon-serializability, and temporal inconsistency [17] to replace the test of view-consistency with less strict tests.

One criticism of the applicability of the method may be that it is based on the premise that the integrity constraints are explicit. We can counter this argument by noting that even in conventional approaches, application programmers must know the integrity constraints for writing consistent transaction programs. In particular, for multidatabases, in most cases, it is safe to ensure that the integrity constraints are made explicit during the integration of the component databases along with other information on the semantics of local systems. Thus, by exploiting knowledge about the constraints we avoid the overheads associated with global serializability. Furthermore, we must note that the enforcement of the above conditions may not require exact knowledge of the constraints. Specifically, for the closure conditions we do not need to know the exact integrity constraints, rather it suffices to know only the set of related data items, for instance that the data item a is constrained by b and c but not necessarily how it is constrained.

As an alternative to system-initiated run-time tests of consistency, the proposed criterion can be realized by writing more *safe* global transaction programs. Transaction programmers should incorporate into their code the possibility of reading inconsistent local data, and for instance, block or exit in such cases. The proposed criterion offers the theoretical basis for the correctness of these tests and defines the exact amount of data that must be considered. For instance from Theorem 2, to ensure correctness, a global transaction program needs to be global-view closed and local-view consistent.

5.2. Isolation

View-based two-level serializability ensures the preservation of database consistency, however, it does not ensure the isolation property for global transactions. In particular, global transactions may read partial results of other global transactions through indirect conflicts with local transactions. Although this may be undesirable for some applications, it may applicable to others such as cooperative transactions.

For transactions that require isolation, we can use the presented criteria in conjunction with methods that enforce global serializability, such as the ticket method [7]. We sketch briefly such an approach. Transactions are divided in two classes, class I and class NI. Class I includes transactions that must be isolated and NI transactions for which isolation may be relaxed. Following the ticket method, all transactions at each local site must read and increment the value of a specific data item, called a ticket. For transactions in NI we also use the view-based approach. Let's assume that optimistic concurrency control is used, where a global serialization graph is build based on the ticket value. Specifically, an edge is added from transaction t_i to transaction t_j, if the ticket value of t_i at a local site is smaller than that of the ticket of t_j at that site. A transaction is validated only if no cycles occur. Using our method, we can ignore cycles among transactions in NI and still maintain database consistency.

5.3. Performance

Closures are computed once at compile, e.g., constraint definition, time. The run time overhead is that of reading closures and testing for consistency. One drawback of the proposed criterion is that it may result in very large readsets for update global transactions as a consequence of imposing closure conditions. The size of closure sets can be reduced if a more elaborate definition of closure is provided as suggested in Section 3, and the time for testing consistency can be minimized if advanced methods for testing consistency are employed [8].

Comparing these overheads to proposed methods of enforcing global serializability [7, 12, 21], our method still avoids creating direct conflicts between global transactions as global serializability methods do by enforcing them to update the same data items. Thus, although readsets may be large, readsets of different transactions may be disjoint, and thus may not cause conflicts. Furthermore, in contrast to global serializability methods, the additional operations appended to global transactions are read operations, which in general allow for more concurrency than write operations. In this respect, the proposed method avoids bottlenecks and long lock waits that may result from forcing conflicting operation in order to achieve global serializability.

Roughly speaking, our method is expected to outperform global serializability when global transactions are independent to each other, in the sense that they read unrelated data. This is exactly the case at which global serializability methods incur their worst overhead by forcing unrelated transactions to conflict although they naturally do not.

6. RELATED WORK

In the previous sections we advanced certain prerequisites to the correctness of 2LSR global schedules. In this section, these conditions will be compared with those advanced in the literature. The correctness of 2LSR global schedules has been examined by Mehrotra et al [13] when no local/global integrity constraints are present and for two multidatabase models, the G_{rw} model, where local transactions are not allowed to read global data and the $G_{rw}L_r$ where local transactions are allowed to read global data. In the G_{rw} model, to avoid inconsistencies, both local and global transaction programs are required to be fixed-structured. A transaction program is *fixed-structured* if its execution from every database state results in transactions with a common structure. In the $G_{rw}L_r$, to avoid inconsistencies, global transaction programs must possess no value dependencies among their global subtransactions. A global subtransaction t_j is *value dependent* on a set of global subtransactions $t_1, ..., t_{j-1}$ if the execution of one or more operations in t_j is determined by the values read by $t_1, ..., t_{j-1}$.

It is illuminating to compare *the range* of acceptable schedules generated by the present work with those encompassed by the above method. Let ST_2LSR denote the set of 2LSR global schedules in which all transactions are fixed-structured; ND_2LSR denote the set of 2LSR global schedules with no value dependencies permitted in global transactions; LV_2LSR denote the set of 2LSR global schedules in which the local views of global transactions are consistent; and LG_2LSR denote the set of 2LSR global schedules in which the local views of global transactions are consistent and the global view of global transactions is closed. Within the G_{rw} model, since ST_2LSR global schedules are correct, the fact that both local and global transactions are fixed-structured implies that their retrievals from local sites will be consistent. However, the possession of consistent local views by global transactions does not imply that both local and global transactions are fixed-structured. Thus, LV_2LSR is a superset of ST_2LSR. Within the $G_{rw}L_r$ model, the fact that a global transaction has no value dependencies does not imply that its retrieval of global data items is closed; nor does the converse hold true. Thus, there is no inclusive relationship between ND_2LSR and LG_2LSR. We now compare further the above conditions in terms of *their applicability* in the multidatabase environment. As pointed out by Mehrotra et al [13] it may be impractical to assume the presence of fixed structured programs, since local transaction programs are pre-existing and may not satisfy these restrictions. Similarly, the prohibition of value dependencies is excessively restrictive, as many applications involve data transfer among different local database sites, resulting in value dependencies among the subtransactions of a global transaction. In contrast, our approach is more practical, since it affects only global transactions and the testing of local view consistency as well as the specifications of global view closures in global transactions can be easily implemented.

Rastogi et al [18] presented additional findings relevant to the present research. That work presented a non-serializable criterion, termed *predicatewise serializability* (PWSR), to be applied in a database environment in which the integrity constraints can be grouped into $C_1 \wedge ... \wedge C_l$, where C_i is defined over a set of data items $d_i \subseteq D$ and $d_i \cap d_j = \emptyset$, $i \neq j$. A schedule is said to be PWSR if, for all i, $i = 1, ..., l$, S^{d_i} is serializable. That research demonstrated that a PWSR schedule S is correct, either if all transaction programs have a fixed-structure or if S is a delayed read schedule. A schedule S is *delayed read* if each transaction t_i in S cannot read a data item written by transaction t_j until the completion of all t_j's operations. This theory may be applied to an MDBS environment in which all local schedules are serializable and either both local and global transactions are fixed-structured or all local schedules are delayed read. Clearly, the present work has advantages over the application of PWSR in the MDBS environment, since PWSR is applicable only if local transactions have a fixed structure or local schedules are delayed-read and there are no local/global integrity constraints.

7. CONCLUSIONS

Enforcing serializability of transaction executions may be restrictive in terms of performance or even inappropriate for some applications. However, by relaxing serializability, the correctness of the database is no longer ensured. The contribution of this paper is twofold. First, we have introduced the concept of view consistency and view closure of transactions. We believe that the relation of these properties of transaction views to integrity constraints provides an innovative approach to maintaining database consistency in the absence of serializability. Second, we have developed a new correctness criterion for multidatabase systems. This new criterion uses the concept of view consistency and view closure, to specify conditions that permit 2LSR global schedules to ensure database consistency. The criterion respects local autonomy, since no restrictions other than serializability need to be imposed on local schedules. We have demonstrated how this criterion can be relaxed for special types of integrity constraints and data access patterns. Finally, we have discussed the feasibility and applicability of the proposed criterion.

Acknowledgements — We would like to acknowledge the anonymous referees for their helpful suggestions and constructive comments on an earlier version of this paper.

REFERENCES

[1] P. Bernstein and B. Blaustein. Fast method for testing quantified relational calculus assertions. In *Proceedings of ACM-SIGMOD International Conference on Management of Data*, pp. 39–50, Orlando, FL (1982).

[2] P. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Databases Systems.* Addison-Wesley (1987).

[3] Y. Breitbart, H. Garcia-Molina, and A. Silberschatz. Overview of multidatabase transaction management. *The VLDB Journal*, **1**(2):181–239 (1992).

[4] W. Du and A. Elmagarmid. Quasi serializability: A correctness criterion for global concurrency control in interbase. In *Proceedings of the 15th International Conference on Very Large Data Bases*, pp. 347–355, Amsterdam, The Netherlands (1989).

[5] A. K. Elmagarmid, editor. *Database Transaction Models for Advanced Applications.* Morgan Kaufmann (1992).

[6] H. Garcia-Molina. Using semantic knowledge for transaction processing in a distributed database. *ACM Transactions on Database Systems*, **8**(2):186–213 (1983).

[7] D. Georgakopoulos, M. Rusinkiewicz, and A. Sheth. Using tickets to enforce the serializability of multidatabase transactions. *IEEE Transactions on Knowledge and Data Engineering*, **6**(1) (1994).

[8] P. Grefen and P. Appers. Integrity control in relational database systems - an overview. *Data and Knowledge Engineering*, **10**:187–223 (1993).

[9] P. Grefen and J. Widom. Integrity constraint checking in federated databases. In *Proceedings of the 1st IFCIS International Conference on Cooperative Information Systems*, pp. 38–47, Brussels, Belgium (1996).

[10] A. Gupta, Y. Sagiv, J. D. Ullman, and J. Widom. Constraint checking with partial information. In *Proceedings of the 13th Symposium on Principles of Database Systems* (1994).

[11] A. Gupta and J. Widom. Local verification of global integrity constraints in distributed databases. In *Proceedings of the ACM-SIGMOD International Conference on Management of Data*, pp. 49–58, Washington, DC (1993).

[12] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. The concurrency control problem in multidatabases: Characteristics and solutions. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp. 288–297 (1992).

[13] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. Maintaining database consistency in heterogeneous distributed database systems. Technical Report TR-91-04, Department of Computer Science, University of Texas at Austin (1991).

[14] S. Mehrotra, R. Rastogi, H. F. Korth, and A. Silberschatz. Non-serializable executions in heterogeneous distributed database systems. In *Proceedings of the 1st International Conference on Parallel and Distributed Information Systems*, pp. 245–252 (1991).

[15] E. Pitoura, O. Bukhres, and A. K. Elmagarmid. Object orientation in multidatabase systems. *ACM Computing Surveys*, **27**(2):141–195 (1995).

[16] C. Pu. Superdatabases for composition of heterogeneous databases. In *Proceedings of the International Conference on Data Engineering*, pp. 548–555 (1988).

[17] K. Ramamritham and P. K. Chrysanthis. A taxonomy of correctness criteria in database applications. *The VLDB Journal*, **5**(1):85–97 (1996).

[18] R. Rastogi, S. Mehrotra, Y. Breitbart, H. F. Korth, and A. Silberschatz. On correctness of non-serializable executions. In *Proceedings of ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pp. 97–108 (1993).

[19] M. Rusinkiewicz, A. Sheth, and G. Karabatis. Specifying interdatabase dependencies in a multidatabase environment. *IEEE Computer*, **24**(12) (1991).

[20] A. Sheth and J. Larson. Federated database systems. *ACM Computing Surveys*, **22**(3):183–226 (1990).

[21] A. Zhang and A. Elmagarmid. A theory of global concurrency control in multidatabase systems. *The VLDB Journal*, **2**(3):331–359 (1993).