10

Correctness Criteria and Concurrency Control

Panos K. Chrysanthis

Department of Computer Science
University of Pittsburgh, Pittsburgh, PA

Krithi Ramamritham

Department of Computer Science
University of Massachusetts, Amherst, MA

Introduction

Database systems have been widely incorporated in almost all aspects of the operations of large organizations. Database systems offer reliability guarantees concerning the correctness of data in spite of failures and concurrent accesses by multiple users. Similar guarantees are expected from *multidatabase systems* (MDBS) which logically integrate multiple pre-existing database systems providing uniform and transparent access to data stored in these databases.

MDBSs respond to the needs of organizations to interoperate their databases already in service supporting new global applications that access multiple databases. An MDBS allows each local database system to continue to operate in an independent fashion. That is, an MDBS preserves the *autonomy* of the local database systems, meaning that the MDBS design does not require any changes to existing databases and applications, or to the local database management systems. Further, they offer flexibility in expanding an existing database platform by adding new local database systems without concerns for compatibility with already existing database systems.

Since different database systems are suitable for different applications, MDBSs are typically heterogeneous employing different correctness criteria and correctness enforcing mechanisms. Correctness criteria define acceptable execution orderings of database operations which satisfy data consistency correctness requirements. Autonomy and heterogeneity cause additional problems and tensions in satisfying traditional correctness criteria in MDBSs; traditional correctness entails control over all data across the multiple databases whereas autonomy implies (1) lack of any such global control and (2) lack of information regarding the execution orderings at the local database systems.

Database correctness has been traditionally studied with respect to the two sources of inconsistencies and incorrectness, namely, concurrency and failures. This chapter focuses on the former discussing the issue of correctness in the absence of failures and the problem of concurrency control in MDBS.

10.1 Databases, Multidatabases and Transactions

A database is an entity that integrates and stores related data in an organized manner. Each database is associated with a set of integrity constraints that captures the correctness of the data in the database. Data values are allowed to be stored in the database, if they do not violate the integrity constraints of the database.

In traditional database systems, correctness of the database is ensured by means of the notion of transactions. A transaction is a program segment in execution that performs a task by accessing and manipulating the database. Transactions are designed so that each preserves the integrity constraints on the database at its boundaries. That is, a transaction takes the database from one consistent state to another only when it executes all its statements. Thus, all the operations of a transaction are treated as a single, indivisible, *atomic* unit and the effects of partially executed transactions are obliterated from the database. The effects of a transaction are also obliterated when the transaction invokes the *abort* transaction management primitive that informs the system that the transaction cannot successfully complete its task. A transaction informs the system that the transaction has successfully completed its task and that all its effects on the objects should be made permanent in the database by invoking the *commit* primitive.

Even if individual transactions are correct programs, transactions can interfere with each other when they are executed concurrently, thereby producing inconsistent results. By inconsistent results, we mean that either the output of the transaction is not consistent with the values of the objects in the database, or the effects of the transaction on the database destroy the integrity of the database. Hence the interleaved execution of a set of transactions is controlled, providing the necessary *isolation* between concurrent transactions to execute without any interferences in observing and operating on a consistent database.

As was mentioned earlier, an MDBS is built on top of a number of existing database systems (Figure 10.1). These database systems are traditional database systems that ensure the atomicity and isolation properties of their transactions.

Two types of transactions execute in an MDBS:

1. *Local transactions* that access data from only a single database.

2. *Global transactions* that access data from multiple databases.

In accordance to the local autonomy requirement, local transactions are submitted directly to the transaction manager (TM) of a local DBMS (LDBMS) and the global transaction manager (GTM) of the MDBS is not aware of their existence. Neither is a LDBMS aware of the existence of global transactions which are submitted directly to the GTM.

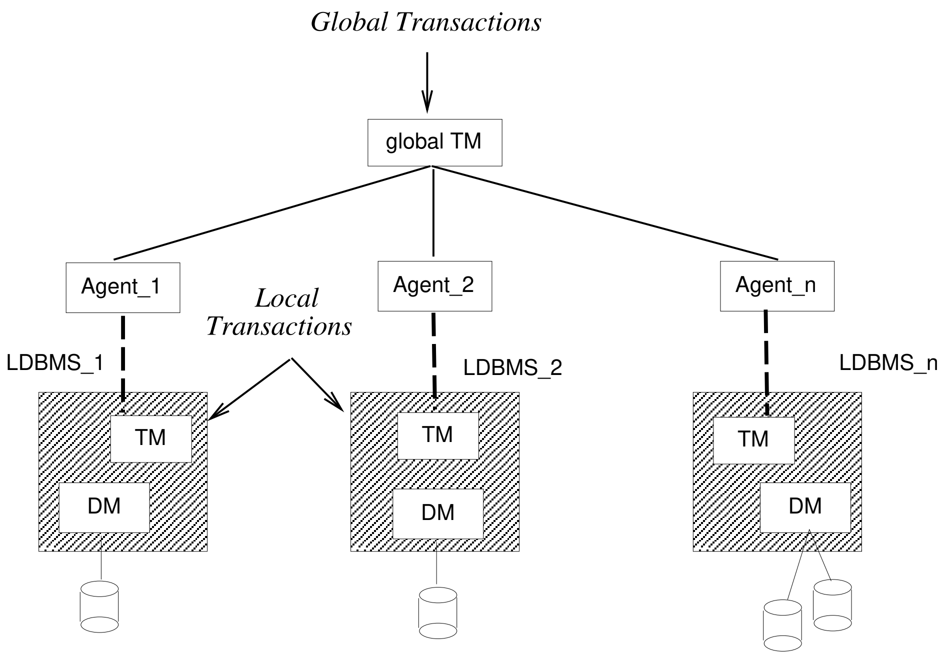

FIGURE 10.1
A Multidatabase System Model

A global transaction G is decomposed by the GTM based on the distribution of the data into several *subtransactions* g_i, each of which executes as a local transaction on some LDBMS with the available data. The GTM submits a substransaction to a LDBMS with the help of an *agent* who resides above each local database system, serving as the interface between the GTM and the local database system.

As TMs are responsible for controlling the execution of local transactions to ensure the correctness of the local database, the GTM is responsible for controlling the execution of global transactions to ensure global consistency. This splitting of control is a manifestation of the tension that exists between autonomy and consistency requirements that complicates correctness in MDBS and makes global concurrency control difficult.

Three main approaches have emerged in addressing the issue of data consistency in MDBS. The first approach attempts to guarantee multidatabase *global serializability* since serializability is a widely used correctness criterion [EH88, WV90, BS88, Pu88, PV88, GRS91, ZE93, ZE93]. This approach also includes proposals for commit protocols suitable for MDBSs [BST90, Geo90, SKS91, AlCh96]. The second approach replaces serializability with other correctness criteria since serializability is considered very constraining when applied to multidatabase environments. In most cases, these correctness criteria are relaxations of serializability, such as *quasi-serializability* [DE89] and *predicate-wise serializability* [KS88]. The third approach re-defines or extends the traditional transaction model to a more suitable transaction model for MDBSs with different correctness properties. These include *sagas* [GS87], *s-transactions* [VE91], *flexible transactions* [RELL90] and *polytransactions* [RSK91, SRK92]. This approach attempts to come up with correctness criteria that view database consistency requirements and transaction correctness properties independently.

We will be using ACTA [CR91, CR94], a first-order logic based formalism, to precisely state correctness requirements, transaction properties, as well as the behavior of transaction processing mechanisms. In the next section we will introduce the ACTA formalism while discussing serializability as the traditional notion of correctness for the concurrent execution.

Whereas serializability and its relaxations are, in general, application and transaction model independent criteria, the other MDBS correctness criteria are application and transaction model specific. In Sections 10.3 and 10.4, we will examine MDBS correctness criteria from these two perspective respectively. Section 10.5 concludes the chapter.

10.2 Correctness of Concurrent Executions

As mentioned above, the execution of a set of transactions need to be controlled so that transactions perform operations on objects in a certain manner that is considered correct. That is, correctness criteria can be defined as constraints on the ordering of concurrent operations and transaction primitives in *histories* [BHG87] which represents the concurrent execution of the set of transactions.

10.2.1 Serializability

Intuitively, any correct execution of a set of transactions must be free from interferences. Assuming that individual transactions are correct, a serial execution of a set of transactions is correct since transactions that execute serially execute in isolation and cannot interfere with each other. *Serializability* is the traditional notion of correctness for the concurrent execution of a set of transactions that satisfies isolation. The key idea is that since serial executions are correct, then any execution whose behavior or effects are the same as some serial execution must also be correct. Such concurrent executions that are equivalent to serial executions are called *serializable*.

In most concurrency control protocols, serializability is based on the notion of conflicting operations and is called *conflict serializability*. Two operations conflict when their effect is order-dependent.

DEFINITION 10.1

Let s be the state of an object representing its value, $return(s, p)$ be the output produced by operation p defined on the object, and $state(s, p)$ be the state produced after the execution of p.

Two operations p and q conflict *in a state s, denoted by* $\mathrm{conflict}(p, q)$, *iff*

$$\begin{aligned}(state(state(s,p),\ q) &\neq state(state(s,q),\ p)) \quad \vee\\ (return(s,\ q) &\neq return(state(s,p),\ q)) \vee\\ (return(s,\ p) &\neq return(state(s,q),\ p)).\end{aligned}$$

Two operations that do not conflict are compatible *or* commute.

That is, two operations conflict if their effects on the state of an object or their return values are not independent of their execution order. Clearly, for this to be true (i.e., the two operations conflict) both should operate on the same object and one of them modifies the object, e.g., it is a write operation. On the other hand, two read operations on the same object commute and can be executed in any order.

Conflict serializability ensures that pairs of conflicting operations appear in the same order in any two equivalent executions [EGLT76, Gra75, Gra78].

DEFINITION 10.2 *Transaction Definition*

Let $p_i[ob]$ be an object event corresponding to the invocation of the operation p on object ob by transaction T_i.

A transaction T_i is a partial order with ordering relation $\rightarrow_i$ where:

1. $T_i = \{p_i[ob] \mid p \text{ is an operation on } ob\} \cup \{\beta_i, \alpha_i, c_i\}$
   Operations in T_i can be operations on objects, begin (β), commit (c) and abort (α) primitives.
2. $\forall i\ \forall p\ \forall ob\ \ p_i[ob] \in T_i \Rightarrow (\beta_i \rightarrow_i p_i[ob])$
   All operations of a transaction follow its begin operation.
3. $\alpha_i \in T_i \Leftrightarrow c_i \notin T_i$
   A transaction can either commit or abort but not both.
4. *if ϵ is c_i or α_i,* $\forall i\ \forall p\ \forall ob\ \ p_i[ob] \in T_i \Rightarrow (p_i[ob] \rightarrow_i \epsilon)$
   All operations of a transaction precede its commit or abort.
5. $\forall i\ \forall p, q\ \forall ob\ p_i[ob] \in T_i \wedge q_i[ob] \in T_i \wedge \mathrm{conflict}(p, q) \Rightarrow$
   $(p_i[ob] \rightarrow_i q_i[ob]) \vee (q_i[ob] \rightarrow_i p_i[ob])$
   The order of conflicting operations is defined.

In general, we use ϵ_j to denote the invocation of an event ϵ by transaction T_j. The predicate $\epsilon \rightarrow_i \epsilon'$ is true if event ϵ precedes event ϵ' in transaction T_i. Otherwise, it is false. (Thus, $\epsilon \rightarrow_i \epsilon'$ implies that $\epsilon \in T_i$ and $\epsilon' \in T_i$.) We will omit the subscript in an ordering relation $\rightarrow$ when its context is clear.

DEFINITION 10.3

Let $T = \{\ T_1, T_2, ..., T_n\ \}$ be a set of transactions. A (complete) history *H of the concurrent execution of a set of transactions T contains all the operations and primitives invoked by the transactions in T and indicates the partial order $\rightarrow_H$ in which these operations occur.*

1. $H = \cup_{i=1}^{n} T_i$
   Operations in H are exactly those in $T_1, ..., T_n$
2. $\forall i\ \forall p, q\ \forall ob\ \ (p_i[ob] \rightarrow_i q_i[ob]) \Rightarrow (p_i[ob] \rightarrow_H q_i[ob])$
   The order of operations within each transaction is preserved.

3. $\forall i, j\ \forall p, q\ \forall ob\ p_i[ob] \in H \wedge q_j[ob] \in H \wedge \text{conflict}(p, q) \Rightarrow$
   $(p_i[ob] \rightarrow_H q_j[ob]) \vee (q_j[ob] \rightarrow_H p_i[ob])$.
   The ordering of every pair of conflicting operations is defined.

H denotes a *complete* history. When a transaction invokes an event, that event is appended to the *current* history, denoted by H_{ct}. That is, H_{ct} is a *prefix* of a complete history, and is useful in capturing incomplete executions of transactions that contain active transactions. Incomplete executions be a result of a system failure or describe a scheduler in execution. A complete history does not contain any active transactions.

The *projection* of a history H according to a given criterion is a subhistory that satisfies the criterion. It is constructed by eliminating any events in H that do not satisfy the given criterion while preserving the partial ordering of the events in the projection. For instance, the projection of a history with respect to committed transactions, denoted by H_{comm}, includes only those events invoked by committed transactions. The projection criterion can be formally specified as

$$\forall t \in T\ \epsilon_t \in H_{comm} \Rightarrow c_t \in H$$

H_{comm} is obtained from H by deleting all operations of non-committed transactions in it. Note that H_{comm} is a complete history with respect to committed transactions in T. Given that in H_{comm} the effects of aborted transactions are nullified, serialization ordering requirements only consider operations invoked by committed transactions, hence serializability criteria are expressed in terms of committed projection of histories.

As mentioned in the previous section, we need to define the notion of equivalence between two histories before being able to determine whether a history is equivalent to a serial history, and hence serializable. Here we will assume that any history H is over a set of committed transactions T_{comm}; i.e., $T = T_{comm}$ and $H = H_{comm}$.

DEFINITION 10.4

Two histories H and H' are (conflict) equivalent, ($H \equiv H'$) if

1. $\forall i\ \forall p\ \forall ob,\ T_i \in T, p_i[ob] \in T_i \quad p_i[ob] \in H \Leftrightarrow p_i[ob] \in H'$
   They are defined over the same set of transactions and have the same operations.
2. $\forall i\ \forall p, q\ \forall ob\ \text{conflict}(p, q) \wedge (p_i[ob] \rightarrow_H q_j[ob]) \Leftrightarrow (p_i[ob] \rightarrow_{H'} q_j[ob])$
   Any pair of conflicting operations p_i and q_j belonging to committed transactions T_i and T_j is ordered in the same way in both histories.

DEFINITION 10.5

A history H_{serial} is serial if for every pair of transactions T_i and T_j that appears in H_{serial}, either all operations of T_i appear before all operations of T_j or vice versa.; i.e.,

$$\exists p_i[ob] \in T_i, q_j[ob] \in T_j \ (p_i \rightarrow_{H_{serial}} p_j) \Rightarrow \\ \forall r_i \in T_i, o_j \in T_j \ (r_i \rightarrow_{H_{serial}} o_j)$$

DEFINITION 10.6

A history H is (conflict) serializable if its committed projection H_{comm} is (conflict) equivalent to a serial history H_{serial}; i.e., $H_{comm} \equiv H_{serial}$.

Clearly, conflicting operations induce serialization ordering requirements between the invoking transactions. Specifically, if an operation $p_i[ob]$ invoked by transaction T_i precedes an operation $q_j[ob]$ invoked by T_j and with which it conflicts (*conflict*(p,q)), then T_i must precede T_j in a serializable history. We will denote this *ordering dependency* by $\mathcal{C}$ binary relation on transactions:

DEFINITION 10.7 *Serialization ordering*

$$\forall t_i, t_j \in T_{comm}, t_i \neq t_j \\ (t_i \ \mathcal{C} \ t_j) \ \textit{iff} \ \exists ob \ \exists p, q \ (\text{conflict}(p, q) \wedge (p_i[ob] \rightarrow q_j[ob]))$$

Let $\mathcal{C}^*$ be the transitive-closure of $\mathcal{C}$; i.e.,

$$(t_i \ \mathcal{C}^* \ t_k) \ \textit{iff} \ [(t_i \ \mathcal{C} \ t_k) \vee \exists t_j \ (t_i \ \mathcal{C} \ t_j \wedge t_j \ \mathcal{C}^* \ t_k)].$$

It has been shown that serializability demands that the serialization ordering must be acyclic [BHG87]. A cycle involving transactions T_i and T_j implies that in any equivalent serial execution T_i should precede T_j and T_j should precede T_i, but both of these cannot hold simultaneously.

DEFINITION 10.8 *Serializability Theorem*

H_{comm} *is* (conflict) serializable *iff* $\forall t \in T_{comm} \ \neg(t \ \mathcal{C}^* \ t)$.

This means that given a history, it can be determined whether it is serializable by testing if its committed projection does not induce any ordering dependency cycles.

A simple way to recognize whether the induced ordering dependencies are acyclic is by constructing a dependency graph, called a *serialization graph* and search for cycles [Gra75, Pap86]. A serialization graph is a directed graph in which nodes represent transactions in H, and an edge $T_i \longrightarrow T_j, i \neq j$, means that T_i has an ordering dependency on T_j, i.e., one of T_i's operations precedes and conflicts with one of T_j's operations in H.

10.3
Application Independent Criteria

In this section, we examine correctness criteria that are applicable to different applications and under different transaction models. We first define the problem of global serializability and then examine different methods for achieving it.

10.3.1 Global Serializability

To ensure global serializability in multidatabase systems, the problem of *indirect conflicts* needs to be overcome. This is illustrated with the following example. Here, $r_i[x]$ ($w_i[x]$) denotes a read (write) operation performed by transaction T_i on item x and c_i denotes the commit primitive of T_i.

EXAMPLE 10.1
Consider two local databases one with items a and b, and another that has c and d.

Consider these transactions:

$G_1 : r_1(a); r_1(c)$
$G_2 : r_2(b); r_2(d)$
$L_3 : w_3(a); w_3(b)$
$L_4 : w_4(c); w_4(d)$

Here G_1 and G_2 are global transactions and L_3 and L_4 are local. Their concurrent execution produces the local histories H_1 and H_2:

$H_1 : r_1(a); c_1; w_3(a); w_3(b); c_3; r_2(b); c_2.$ $\quad G_1 \rightarrow L_3 \rightarrow G_2$
$H_2 : w_4(c); r_1(c); c_1; r_2(d); c_2; w_4(d); c_4.$ $\quad G_2 \rightarrow L_4 \rightarrow G_1$

This example shows that even a serial execution of global transactions does not guarantee global serializability! This is because local transactions can create *indirect* conflicts between global transactions.

This problem of indirect conflicts is better understood by formalizing global serializability and identifying the sources of the conflicts.

Let MDBS consist of a set of databases $\{D_1, D_2, ... D_n\}$ where D_i is managed by LDBMS$_i$.

Let G be the set of global transactions and g_n^s be a (sub)transaction of a global transaction g_n ($g_n \in G$) executing all the operations of g_n on site s.

Let T_s be the set of transactions, both local transactions and global (sub)transactions, executing on site s. $T = (\cup_s T_s)$.

Let $L_s = \{l_1^s, l_2^s, .., l_n^s\}$ be the set of local transactions executing on site s ($L_s \subset T_s$).

Let H be the global history of events in MDBS relating to committed transactions in T. H is a set of local histories $H = \{H_1, H_2, .., H_n\}$ where H_s is the history defined over T_s.

Given the two types of transactions in MDBSs, the definition of transitive-closure of $\mathcal{C}$ which is used in the characterization of serializability (definition 10.8), can be refined.

DEFINITION 10.9 *Local Indirect Conflicts*

Let $\mathcal{I}$ be a binary relation on a set of transactions T_s. $\forall\, t_x,\ t_y\ \in\ T_s$,

$$(t_x\ \mathcal{I}\ t_y) \text{ if}$$
$$\exists k > 1, t_0 = t_x, t_k = t_y\ \forall i, 1 \le i \le k-1\ t_i \in T_s$$
$$(t_{i-1}\ \mathcal{C}\ t_i) \wedge (t_i\ \mathcal{C}\ t_{i+1})$$

where $\mathcal{C}$ is the binary relation defined in Definition 10.7 that captures direct conflicts between two transactions.

The $\mathcal{I}$ relation captures the fact that two transactions, local or global subtransactions, executing on the same site might *indirectly conflict* in a local history even if they do not access any shared objects. The latter is made clear when $\mathcal{C}$ is substituted with its definition.

DEFINITION 10.10

Let $\mathcal{I}$ be a binary relation on a set of transactions T_s. $\forall\, t_x,\ t_y\ \in\ T_s$,

$$(t_x\ \mathcal{I}\ t_y) \text{ if}$$
$$\exists k > 1, t_0 = t_x, t_k = t_y\ \forall i, 1 \le i \le k-1\ t_i \in T_s$$
$$\exists ob\ \exists p, q\ ((\text{conflict}(p_{t_{i-1}}[ob], q_{t_i}[ob]) \wedge (p_{t_{i-1}}[ob] \rightarrow q_{t_i}[ob])) \wedge$$
$$\exists ob'\ \exists p', q'\ (\text{conflict}(p'_{t_i}[ob'], q'_{t_{i+1}}[ob']) \wedge (p'_{t_i}[ob'] \rightarrow q'_{t_{i+1}}[ob'])))$$

DEFINITION 10.11 *Remote Indirect Conflicts*

Let $\mathcal{R}$ be a binary relation on a set of transactions T_s. $\forall\, t_x,\ t_y\ \in\ T_s$,

$$(t_x\ \mathcal{R}\ t_y) \text{ if}$$
$$\exists k \ne s, g_i^k, g_j^k\ \ [(t_x\ \mathcal{C}\ g_i^s) \vee (t_x\ \mathcal{I}\ g_i^s)] \wedge$$
$$[(g_i^k\ \mathcal{C}\ g_j^k) \vee (g_i^k\ \mathcal{I}\ g_j^k) \vee (g_i^k\ \mathcal{R}\ g_j^k)] \wedge$$
$$[(g_j^s\ \mathcal{C}\ t_y) \vee (g_j^s\ \mathcal{I}\ t_y)]$$

The $\mathcal{R}$ relation captures the fact that two transactions, local or global subtransactions, executing on the same site might *indirectly conflict* in a global history due to conflicts between subtranctions of the same global transaction executing on different sites.

By expressing $\mathcal{C}^*$, the transitive-closure of $\mathcal{C}$, specifically for MDBSs in terms of local indirect conflicts $\mathcal{I}$ and of remote indirect conflicts $\mathcal{R}$, global serializability in MDBS can be characterized as follows:

DEFINITION 10.12

H *is* global serializable *iff* $\forall\ t\ \in\ T\ \neg(t\ \mathcal{I}\ t) \wedge \neg(t\ \mathcal{R}\ t)$

Since local transactions execute under the control of individual LDBMSs and global transactions under the control of GTM, global serializability is achieved if each LDBMS$_i$ ensures that $\forall l_i^s \in T, \neg(l_i^s\ \mathcal{C}^* l_i^s)$ and that GTM ensures that $\forall g_i^s \in T, \neg(g_i^s\ \mathcal{C}^* g_i^s)$. A LDBMS can always recognize local indirect conflicts. But an LDBMS can never recognize remote indirect conflicts. On the other hand, GTM can recognize both local and remote indirect conflicts as long as no local transactions are involved. Thus, with no knowledge available about the internals of LDBMSs, without any exchange of control information, or some restrictions on the executions of operations to curtail indirect conflicts, global serializability cannot be achieved in MDBS by tracing and testing for the acyclicity of ordering dependencies.

10.3.2 Achieving Global Serializability

All current schemes for achieving global serializability assume that (1) all LDBMS ensure serializability and (2) no control information can be exchanged between the GTM and LDBMSs. However, these schemes can be classified based on whether or not they assume *unlabeled LDBMS* – no knowledge is available about the internals of a LDBMS, for example, if it uses locking or timestamp based concurrency control.

The ticket method [Geo90, GRS91] is an example of a scheme that applies to unlabeled databases. Here conflicts are forced among global transactions using a data item called the *ticket*, maintained at each local database. Global transactions must read the ticket value, increment it, and write the incremented value into the database. By doing so, any remote indirect conflicts between two local transactions at a site are converted into local indirect conflicts (in Definition 10.11 of $\mathcal{R}$, if k=s then $\mathcal{R}$ becomes equivalent to $\mathcal{I}$) that can be detected by the LDBMS, hence LDBMSs can ensure serializability of their local transactions in the presence of global subtransactions. At the same time, GTM does not have to be concerned with indirect conflicts between global subtransactions involving local transactions since all global subtransactions at a site conflict directly. GTM has to only be concerned with remote indirect conflicts involving only global subtransactions to ensure global serializability. In other words, GTM has to make sure that the execution ordering of global subtransactions is consistent across all sites.

GTM achieves consistent execution ordering of global subtransactions across sites using the values of tickets returned by each subtransaction. The idea is similar to timestamp ordering that prevents cyclic execution orderings. The subtransactions of two global transactions execute in a consistent way in all the common sites, if the ticket values of all the subtransactions of one of the transactions, say g_i, are less than the corresponding ones of the other transaction, say g_j; i.e., $\forall s \; ticket(g_i^s) < ticket(g_j^s)$. Thus, GTM aborts a global transaction if one of its subtransactions returns a ticket that violates any previously established ticket ordering between pairs of global transactions. Ticket validation can be either performed in a pessimistic or optimistic fashion.

Several solutions for achieving global serializability are available if we know something about the local concurrency control schemes and in particular their ordering properties. For example, a general scheme for *labeled LDBMS* is based on the notion of *serialization point (SP)* – a distinguished action that determines the serialization order of the transaction in a local history. Serialization points include:

- TO (Timestamp Ordering) – Assignment of timestamp.
- 2PL (Two-Phase Locking) – SP at first lock release.

This notion can be used to enforce global consistency. Essentially, each site s_k is viewed as a single data object o_k and two serialization point actions $sp_i(o_k)$ and $sp_j(o_k)$ always conflict.

Schemes can also be developed based on the recoverability properties of local schedules. A strongly recoverable DBMS restricts SP's to occur at the end of a transaction [BGRS91, Raz92]. A schedule is strongly recoverable if g_i has a direct conflict with g_j, g_j does not commit before g_i commits. The GTM avoids cycles by making sure global transactions do not overlap – by ensuring that commit processing is done serially. That is, between the time a g_i issues its first commit at a site and its last commit at another site, no other g_j issues any commits.

If local databases produce *rigorous* schedules [BGRS91], say they use the strict 2PL protocol [EGLT76, BHG87], then our previous example of Section 10.3.1 cannot occur. Thus, if every local DBMS generates a rigorous schedule, then the GTM does not perform any operation coordination, besides avoiding to commit any global subtransaction at a site before all the operations of the global transaction have been completed.

In summary, we have a hierarchy of local schedule classes, going from most general to most restrictive:

- Serializable
- Strongly Serializable: Coordinate execution of all operations of global transactions so that global transactions execute serially.
- SP-schedules: Coordinate transaction serialization point operations.
- Strongly Recoverable: Coordinate commit operations to execute serially.
- Rigorous: No GTM coordination across global transactions. Defer commit of global subtransactions until the end of the global transaction.

As the local scheduler becomes more restrictive, GTM can be more permissive in coordinating operations of global transactions.

10.3.3 Achieving Relaxations of Global Serializability

Rather than using serializability, alternative consistency notions are also possible. The basic idea is to consider serializability at two levels by considering two types of data, *local* and *global*, and three types of constraints:

- *Local*: Constraints involving only local items (at a single site).
- *Global*: span more than one site, involve only global data items.
- *Global/Local*: Single site constraints, but involve both local and global items.

A local transaction not modify global data and the global layer is normally "added on top".

A global schedule S is *two-level serializable* (2LSR) if it is locally serializable and its projection to a set of global transactions is serializable [MRKS91]. Hence, 2LSR schedules are not always serializable. But they preserve global constraints regardless of the state of local data items. Another possibility is to constrain the access patterns of transactions such that there is no possibility of interactions between local and global transactions.

DEFINITION 10.13

H is two-level serializable *iff*

1. $\forall s \; \forall t \in L_s \; \neg(t \; \mathcal{C}^* \; t)$, *and*
2. $\forall g \in G \; \neg(g \; \mathcal{C}^* \; g)$.

Recall G is the set of global transactions and L_s is the set of local transactions, executing on a site s.

We now examine several other relaxations, including cooperative serializability, multidatabase serializability, and quasi serializability.

Under *cooperative serializability* (CoSR) [Chr91, RC96], transactions form cooperative transaction sets. A cooperative transaction set could be formed by the components of an extended transaction or transactions collaborating over some objects while maintaining the consistency of the objects. In such cases, consistency can be maintained if other transactions which do not belong to the set are serialized with respect to all the transactions in the set. In other words, a set of cooperative transactions becomes the unit of concurrency with respect to serializability.

Let T_c be a set of cooperative transactions, $T_c \subseteq T$.

Let $\mathcal{C}_c$ be a binary relation on transactions in T.

Let H be the history of events relating to committed transactions in T.

DEFINITION 10.14 *Cooperative serialization ordering*

$$
\begin{array}{l}
\forall\; t_i,\; t_j,\; t_k \;\in\; T, t_i \neq t_j, t_i \neq t_k, t_j \neq t_k \;\; \forall T_c \subseteq T \\
(t_i \; \mathcal{C}_c \; t_j),\ \mathit{if}\ \exists ob\ \exists p, q \\
\quad (((t_i \notin T_c \vee t_j \notin T_c) \;\wedge (\text{conflict}(p_{t_i}[ob], q_{t_j}[ob]) \wedge (p_{t_i}[ob] \rightarrow q_{t_j}[ob]))) \vee \\
\quad (t_i \notin T_c, t_j \in T_c, t_k \in T_c \; (\text{conflict}(p_{t_i}[ob], q_{t_k}[ob]) \wedge (p_{t_i}[ob] \rightarrow q_{t_k}[ob]))) \vee \\
\quad (t_i \in T_c, t_j \notin T_c, t_k \in T_c \; (\text{conflict}(p_{t_k}[ob], q_{t_j}[ob]) \wedge (p_{t_k}[ob] \rightarrow q_{t_j}[ob]))))
\end{array}
$$

In this definition, the first clause expresses how a dependency between two transactions which do not belong to the same set is directly established when they invoke conflicting operations on a shared object. This is similar to the clause in the classical definition of (conflict) serializability (definition 10.7). The other two clauses reflect the fact that when a transaction establishes a dependency with another transaction, the same dependency is established between all the transactions in their corresponding cooperative transactions sets. These clauses can be viewed as expressions of the development of dependencies between transaction sets.

DEFINITION 10.15

H *is* cooperative serializable *iff* $\forall t \in T \; \neg(t \; \mathcal{C}_c^* \; t)$.

Note that if each T_c is a singleton set, then no cooperation occurs and $\mathcal{C}_c$ is equivalent to $\mathcal{C}$. In addition, cooperative serializability does not imply that all transactions in a cooperative set must commit or none. For example, let us consider the notion of *multidatabase serializability or m-serializability (MSR)* [MRKS91, MRBKS92] which is another correctness criterion proposed for transactions in multidatabase systems.

Specifically, MSR is defined in the context of emulating *two-phase commit* (2PC) protocols in multidatabases using redo transactions. The idea is that the commitment of a global transaction can be decided using the 2PC protocol between the multidatabase agents that interface the local DBMS without the participation of the local DBMS and hence, a subtransaction is not required to enter the prepare to commit state during the decision phase. If the subtransaction is aborted but the final decision is to commit the global transaction, the updates of the aborted subtransaction are performed subsequently by a redo transaction. This implies that (1) the state of the database against which the redo transaction executes, should be the same as the one seen by the aborted subtransaction and (2) the redo transaction should not invalidate any other active or committed (sub)transaction. In a MSR local schedule, although a subtransaction g_i and its redo transaction $Redo(g_i)$ execute as independent transactions, they are considered together as a pair. That is, database consistency is preserved by serializing all other transactions executing on the same node with respect to the pair $\{g_i, Redo(g_i)\}$. Such a pair is an instance of cooperative transactions and the history H of interest includes events associated with all transactions, i.e., both committed and aborted transactions. MSR then corresponds to cooperative serializability if all conflicts in this history are considered with respect to two types of cooperative transaction sets: $\{g_j\}$ in case $\{g_j\}$ commits and $\{g_i, Redo(g_i)\}$ in case $\{g_i\}$ aborts.

Quasi Serializability has been proposed in [DE89] as a correctness criterion for maintaining transaction consistency in multidatabases. As mentioned earlier, transactions in these systems are either local, i.e., execute on a single site, or global, i.e., execute on multiple sites. QSR assumes that at most one (sub)transaction of a global transaction executes on a particular site.

In QSR, the correctness of the execution of a set of global and local transactions is based on the notion of a *quasi serial history* which, unlike a serial history, specifies that only global transactions are executed serially. A history is *quasi serial* if (1) all local histories are (conflict) serializable, and (2) there exists a total order of all global transactions g_m and g_n where g_m

precedes g_n in the order and all g_m's operations precede g_n's operations in all local histories in which they both appear. A quasi serializable history is equivalent to a quasi serial history.

Indirect conflicts between local transactions induced by conflicts of global transactions that execute on multiple sites are not captured by QSR, the reason being that QSR assumes *no* data dependency across sites. That is, remote direct conflicts captured by $\mathcal{R}$ (definition 10.11) are not considered.

DEFINITION 10.16

H is quasi serializable *iff*

1. $\forall s\ \forall t \in T_s\ \neg(t\ \mathcal{C}^*\ t)$, *and*
2. $\forall g \in G\ \neg(g\ \mathcal{R}^*\ g)$.

It is also appropriate to view QSR in terms of cooperative serializability. Specifically, transactions executing in each site s form a cooperative transaction set, with conflict relation $\mathcal{C}$ applied to them where as global transactions form another cooperative transaction set with $\mathcal{C}$ again being the conflict relation applied to them.

The *Site-graph* protocol [BS88] and *Altruistic Locking* protocol [SGS94] are among the first methods proposed to achieve global serializability. However, it has been shown that both of them produce histories that are QSR.

- Site-graph algorithm: Here each GTM maintains a bipartite graph – (transactions and sites as the nodes). An arc is entered connecting T_i to each site the node will run at. T_i will run if it is not involved in any cycles. Thus, serial execution occurs if sites intersect.

- Altruistic Locking: Here each site has a lock maintained by the global transaction manager. If a transaction can obtain all the locks for the sites that it will visit, it proceeds, otherwise it waits. However, transactions can release their site locks once they are done with a site even if the transaction is still in progress at the other sites. In this case, a transaction that obtains the lock executes in the *wake* of the lock releasing transaction, as in the case of the original altruistic approach. That is, the second transaction is serialized after the first.

It should be noted that in the case of labeled LDBS as rigorous, QSR is equivalent to global serializability and altruistic locking and the site-graph protocols can be used to guarantee global serializability.

10.4 Application Dependent Criteria

In this section we discuss several criteria whose applicability is restricted to a class of applications.

Consider Epsilon Serializability (ESR) [PA91, WYP92], a generalization of classic serializability (SR), explicitly allows some limited amount of inconsistency in transaction processing (TP). ESR is mainly applicable to numeric data. ESR enhances concurrency since some non-SR execution schedules are permitted. For example, epsilon-transactions (ETs) that just perform queries execute in spite of ongoing concurrent updates to the database. Thus, the query ETs view uncommitted, i.e., possibly inconsistent, data. Concretely, an update transaction *export* some inconsistency when it updates a data item while query ETs are in progress. Conversely, a query ET *import* some inconsistency when it reads a data item while uncommitted updates on that data item exist. The correctness notion in ESR is based on bounding the amount of imported and exported inconsistency for each ET [RP95]. The benefits of ESR have been discussed in the papers cited above. For instance, ESR may increase system availability and autonomy [PA91] in distributed TP systems since asynchronous execution is allowed.

Interdependent data are data stored in separate component databases which have mutual consistency requirements and whose relationship is established with the formation of the MDBS. Replicated or derived data are examples of interdependent data. Interdependent data are accessible to both global and local transactions.

So far we have approached the issue of consistency from the perspective of global transactions. Consistency, from the perspective of local transactions which access data at only a local database, is expected to be handled by the LDBMS according to the autonomy assumption. However, this assumes, in addition, that either local transactions do not access interdependent data or there are no interdependent data stored in different databases. Since these are not realistic assumptions we now consider the issue of maintaining the consistency of interdependent data accessed by local transactions.

Rusinkiewski, Sheth and Karabatis have proposed the concept of *poly-transactions* by which the system, instead of re-programming local transactions as global transactions, generates related updates to restore the consistency of interdependent data [RSK91]. The generation of the required updates and their execution strategy as transactions are driven by the *interdatabase dependency schema*, a declarative specification of interdatabase dependencies.

The interdatabase dependency schema is a group of *data dependency descriptors* $\mathrm{D}^3 =< S, U, P, C, A >$, where S is the set of source data objects and U is the target data object. P is a predicate specified using operators of relational algebra that captures the relationship between the source and target data objects. C is a boolean-valued mutual consistency predicate that specify consistency requirements and defines when P is satisfied in terms of time (temporal consistency) and/or state (spatial consistency). Finally A is a collection of consistency restoration actions that are required to restore consistency and to ensure that P is satisfied. That is, if a transaction updates a data object that is related to data objects in other databases and causes a violation of consistency requirement, a series of transactions corresponding to the restoration actions are initiated by the system.

Dependency descriptors, by means of C and A, can provide more semantic information needed for consistency maintenance than database integrity constraints. They support execution of multidatabase transactions resembling that of triggers in advanced active databases allowing actions to perform on the basis of either the state of data or external events.

Consider two sites in a multidatabase which respectively maintain two versions of a data item, one which was complete and another which only contains data required at that site. The two are not required to be consistent at all times but changes done to the complete database are required to percolate to the other within a specified delay. Clearly, this type of constraint is related to deferred consistency restoration. If the changes should be reflected within d units of time, we have the following "temporal commit dependency":

$$({}_{t_i} \in H \wedge (ts({}_{t_i}) = t)) \Rightarrow {}_{t_j} \in H^{t+d}.$$

This says that if t_i commits at time t, t_j should commit *by* time $t + d$.

Sheth and Rusinkiewicz [SR90] have proposed *eventual consistency*, similar to identity connections introduced by Wiederhold and Qian [WQ87], and *lagging consistency*, similar to asynchronously updated copies like quasi-copies [ABG90]. Barbara and Garcia-Molina [BaGM90] have proposed *controlled inconsistency* which extends the work on quasi-copies [ABG90]. Their demarcation protocol [BaGM92] that handles linear inequality constraints, can be used for implementing ESR in distributed transaction processing systems. Recently, the demarcation protocol is generalized in [MY98] to handle more complex constraints by observing that the key idea is to find local sufficient conditions for a global constraint.

Thus far, we have discussed correctness criteria assuming that both local and global transactions are traditional transactions associated with the ACID properties [Gra81, HR83]: *(i) atomicity*, *(ii) consistency*, *(iii) isolation* and *(iv) durability*. Given that serializability is a very restrictive correctness criterion and ensuring global serializability as well as atomicity in MDBS is a difficult task, a number of new transaction models that re-define or extend the traditional transaction model have been proposed as more suitable for MDBSs. These extended transaction models exploit the semantics of the applications and the characteristics of the individual LDBSs, while at the same time consider the limitations in processing global transactions due to the autonomy requirement.

A common characteristic of these transaction models is that they exhibit an hierarchical structure where transactions at any level of the hierarchy unilaterally release their updates in the database. These includes sagas [GS87], Flex transactions [BEK93], s-transactions [VE91], DOM transactions [Buch92], and the ConTract Model [WR92] (see relevant chapter in this book).

10.5 Conclusions

How to manage and guarantee consistency of semantically related data in heterogeneous and distributed computing environments is an issue that involves tradeoffs between autonomy and correctness requirements. Autonomy is an issue because substantial capital investments have been made in local DBMSs and from a practical point of view, requiring major modifications not be an option. Thus, for legacy systems, some weakening of correctness requirements, *vis. a vis.*, serializability becomes unavoidable. Full data consistency and serializability can only be achieved with severe restrictions, in the form of proscriptions and prescriptions [ChRa94] with respect to data access, transaction processing, and commit protocols. Furthermore, even if we are able to achieve global serializability, there are performance implications. Consider for instance the ticket method. It forces conflicts between transactions and so runs the risk of forced conflicts even when there is no potential for direct or indirect conflicts to start with.

Fortunately, emerging standards demand a 2PC protocol to be supported by transaction processing systems that will allow for a control of the commitment of global transactions, simplifying transaction management in an MDBS. Nevertheless, as long as some of the black boxes that join a multidatabase environment do not even support transactional computations, some form of support is needed either from the agents that sit above the black boxes or from the application programmers.

Bibliography

[AlCh96] Al-Houmaily Y. J. and P. K. Chrysanthis. Dealing with Incompatible Presumptions of Commit Protocols in Multidatabase Systems. In *Proceedings of the 11th ACM Annual Symposium on Applied Computing*, pages 186–195, 1996.

[ABG90] Alonso R., D. Barbara and H. Garcia-Molina. Data Caching Issues in an Information Retrieval Systems. *ACM Transactions on Database Systems*, Vol. 15, No. 3, pages 359–384, 1990.

[ANRS92] Ansari M., L. Ness, M. Rusinkiewicz and A. Sheth. Using Flexible Transactions to Support Multi-system Telecommunication Applications. In *Proceedings of the 18th International Conference on Very Large Databases*, pages 65-76, 1992.

[ASSR93] Attie P., M. Singh, A. Sheth and M. Rusinkiwicz. Specifying and Enforcing Intertask Dependencies. In *Proceedings of the 19th International Conference on Very Large Databases*, pages 134–143, 1993.

[BaGM90] Barbara D. and H. Garcia-Molina. The Case for Controlled Inconsistency in Replicated Data. In *Proceedings of the Workshop on the Management of Replicated Data*, pages 35–42, 1990.

[BaGM92] Barbara D. and H. Garcia-Molina. The Demarcation Protocol: A Technique for Maintaining Arithmetic Constraints in Distributed Database Systems. In *Proceedings of the International Conference on Extending Data Base Technology*, pages 371–397, 1992.

[BHG87] Bernstein, P. A., Hadzilacos, V., and Goodman, N. *Concurrency Control and Recovery in Database Systems.* Addison-Wesley, Reading, MA, 1987.

[Breit90] Breitbart Y. et al. Final Report of the Workshop on Multidatabases and Semantic Interoperability. *Technical Report*, Tulsa, OK, 1990.

[BGS92] Breitbart Y., H. Garcia-Molina and A. Silberschatz. Overview of Multidatabase Transaction Management. *VLDB Journal*, Vol. 1, No. 2, pages 181–293, 1992.

[BGRS91] Breitbart Y., D. Georgakopoulos, M. Rusinkiewicz and A. Silberschatz. On Rigorous Transaction Scheduling, *IEEE Transactions on Software Engineering*, Vol. 17, No. 9, pages 954–960, 1991.

[BS88] Breitbart Y. and A. Silberschatz. Multidatabase Update Issues. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 135–142, 1988.

[BST90] Breitbart Y., A. Silberschatz and G. Thompson. Reliable Transaction Management in a Multidatabase System. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 215–224, 1990.

[Buch92] Buchmann A. et al. A Transaction Model for Active Distributed Object Systems. In Elmagarmid, A. K. (Ed.), *Database Transaction Models for Advanced Applications*, pages 123–158, Morgan Kaufmann, 1992.

[BEK93] Bukhres O., A. Elmagarmid and E. Kuhn. Implementation of the Flex Transaction Model. *IEEE Data Engineering*, Vol. 16, No. 2, pages 28–32, 1993.

[Chr91] Chrysanthis, P. K. *ACTA, A Framework for Modeling and Reasoning about Extended Transactions.* PhD thesis, Department of Computer and Information Science, University of Massachusetts, Amherst, Massachusetts, 1991.

[CR91] Chrysanthis, P. K. and Ramamritham, K. A Formalism for Extended Transaction Models. *Proceedings of the 17th International Conference on VLDB*, pages 103–111, 1991.

[CR92] Chrysanthis P. K. and K. Ramamritham. ACTA: The SAGA Continues. In Elmagarmid, A. K. (Ed.), *Database Transaction Models for Advanced Applications*, pages 349–398, Morgan Kaufmann, 1992.

[CR94] Chrysanthis P. K. and K. Ramamritham. Synthesis of Extended Transaction Models using ACTA. *ACM Transactions on Database Systems*, Vol. 19, No. 3, pages 450–491, 1994.

[ChRa94] Chrysanthis P. K. and K. Ramamritham. Autonomy Requirements in Heterogeneous Distributed Database Systems. In *Proceedings of the Conference on the Advances on Data Management*, pages 283-302, 1994.

[DSW94] Deacon A., H. J. Schek and G. Weikum. Semantics-based Multilevel Transactions Management in Federated Systems. In *Proceedings of the 10th International Conference on Data Engineering*, pages 452–461, 1994.

[DE89] Du W. and A. K. Elmagarmid. Quasi Serializability: a Correctness Criterion for Global Concurrency Control in InterBase. In *Proceedings of the 15th International Conference on Very Large Databases*, pages 347–355, 1989.

[Elm92] Elmagarmid A. K. (Ed.) *Database Transaction Models for Advanced Applications.* Morgan Kaufmann, 1992.

[EH88] Elmagarmid, A. K. and Helal, A. A. Supporting Updates in Heterogeneous Distributed Database Systems. In *Proceedings of the IEEE Fourth International Conference on Data Engineering*, 1988.

[ELWR90] Elmagarmid A. K., Y. Leu, W. Litwin and M. Rusinkiewics. A Multidatabase Transaction Model For InterBase. In *Proceedings of the 16th International Conference on Very Large Databases*, pages 507–518, 1990.

[ELLR90] Elmagarmid, A., Leu, Y., Litwin, W., and Rusinkiewicz, M. A Multidatabase Transaction Model for InterBase. In *Proceedings of the 16th International Conference on VLDB*, pages 507–518, 1990.

[EGLT76] Eswaran, K., Gray, J., Lorie, R., and Traiger, I. The Notion of Consistency and Predicate Locks in a Database System. *Communications of the ACM*, 19(11):624–633, 1976.

[GGK+91] Garcia-Molina, H., Gawlick, D., Klein, J., Kleissner, K., and Salem, K. Modeling Long-Running Activities as Nested Sagas. *Bulletin of the IEEE Technical Committee on Data Engineering*, 14(1):14–18, 1991.

[GS87] Garcia-Molina, H. and Salem, K. SAGAS. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 249–259, 1987.

[Geo90] Georgakopoulos D. *Transaction Management in Multidatabase Systems.* Ph.D. Thesis. Department of Computer Science, University of Houston, Houston, Texas, 1990.

[GRS91] Georgakopoulos D., M. Rusinkiewicz and A. Sheth. On Serializability of Multidatabase Transactions through Forced Local Conflicts. In *Proceedings of the 7th International Conference on Data Engineering*, pages 314–323, 1991.

[Gra78] Gray J. Notes on Data Base Operating Systems. In Bayer R., R. M. Graham and G. Seegmuller (Eds.), *Operating Systems: An Advanced Course*, *Lecture Notes in Computer Science*, Volume 60, pages 393–481, Springer-Verlag, 1978.

[Gra81] Gray, J. The Transaction Concept: Virtues and Limitations. In *Proceedings of the 7th International Conference on VLDB*, pages 144–154, 1981.

[Gra75] Gray J. N., R. A. Lorie, A. R. Putzulo and I. L. Traiger. Granularity of Locks and Degrees of Consistency in a Shared Database. In *Proceedings of the First International Conference on Very Large Databases*, pages 25–33, 1975.

[GR93] Gray J. N. and A. Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann, 1993.

[HR83] Haerder T. and A. Reuter. Principles of Transaction-Oriented Database Recovery. *ACM Computing Surveys*, Vol. 15, No. 4, pages 287–317, 1983.

[Kai90] Kaiser G. E. A Flexible Transaction Model for Software Engineering. In *Proceedings of the 6th International Conference on Data Engineering*, pages 560–567, 1990.

[KR93] Kamath M. and K. Ramamritham. Performance Characteristics of Epsilon Serializability with Hierarchical Inconsistency Bounds. In *Proceedings of the 9th International Conference on Data Engineering*, pages 587–594, 1993.

[KS88] Korth, H. F. and Speegle, G. Formal Models of Correctness without Serializability. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 379–386, 1988.

[MY98] Mazumdar, S. and G. Yuan. Localizing Global Constraints: A Geometric Approach. *Proceedings of the 9th International Conference on Computing and Information*, 1998.

[MRBKS92] Mehrotra S., R. Rastogi, Y. Breitbart, H. Korth and A. Silberschatz. The Concurrency Control Problem in Multidatabase Systems: Characterization and Solutions. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 288–297, 1992.

[MRKS91] Mehrotra S., R. Rastogi, H. Korth and A. Silberschatz. Non-serializable Executions in Heterogeneous Distributed Database Systems. In *Proceedings of the First International Conference on Parallel and Distributed Information Systems*, 1991.

[Pap86] Papadimitriou, C. H. *The Theory of Database Concurrency Control.* Computer Science Press, 1986.

[PV88] Pons J. and J. Vilarem. Mixed Concurrency Control: Dealing with Heterogeneity in Distributed Database Systems. In *Proceedings of the 14th International Conference on Very Large Databases*, pages 445–456, 1988.

[Pu88] Pu C. Superdatabases for Composition of Heterogeneous Databases. In *Proceedings of the 4th International Conference on Data Engineering*, 1988.

[PKH88] Pu, C., Kaiser, G., and Hutchinson, N. Split-Transactions for Open-Ended activities. In *Proceedings of the 14th International Conference on VLDB*, pages 26–37, 1988.

[PA91] Pu C. and A. Leff. Replica Control in Distributed Systems: An Asynchronous Approach. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 377–386, 1991.

[RC93] Ramamritham K. and P. K. Chrysanthis. In Search of Acceptability Criteria: Database Consistency Requirements and Transaction

Correctness Properties. In Ozsu, Dayal and Valduriez (Eds.), *Distributed Object Management*, Morgan Kaufmann Publishers, 1993.

[RC96] Ramamritham K. and P. K. Chrysanthis. A Taxonomy of Correctness Criteria in Database Applications. *Journal of Very Large Databases*, Vol. 4, No. 1, pages 181-293, 1996.

[RP95] Ramamritham K. and C. Pu. A Formal Characterization of Epsilon Serializability. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 7, No 6, pages 997–107, 1995.

[Raz92] Raz Y. The Principles for Commit Ordering, or Guaranteeing Serializability in a Heterogeneous Environment of Multiple Autonomous Resource Managers using Atomic Commitment. In *Proceedings of the 18th International Conference on Very Large Databases*, 1992.

[RW91] Reuter A. and H. Wächter. The ConTract Model. *IEEE Data Engineering*, Vol. 14, No. 1, pages 39–43, 1991.

[RELL90] Rusinkiewicz M., A. Elmagarmid, Y. Leu and W. Litwin. Extending the Transaction Model to Capture More Meaning. *ACM SIGMOD Record*, Vol. 19, No. 1, pages 3–7, 1990.

[RSK91] Rusinkiewicz M., A. Sheth and G. Karabatis. Specifying Interdatabase Dependencies in Multidatabase Environment. *IEEE Computer*, Vol. 12, No. 12, pages 46–54, 1991.

[SGS94] Salem K., H. Garcia-Molina and J. Shands. Altruistic Locking. *ACM Transactions on Database Systems,* Vol. 19, No. 1, pages 117–165, 1994.

[Sch93] Schwenkreis F. APRICOTS: A Prototype Implementation of a ConTract-System Managing of the Control-Flow and the Communication-System. In *Proceedings of 12th Symposium on Reliable Distributed Systems,* pages 12–22, 1993.

[SL90] Sheth A. and J. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys*, Vol. 22, No. 3, pages 183–236, 1990.

[SR90] Sheth A. and M. Rusinkiewicz. Management of Interdependent Data: Specifying Dependency and Consistency Requirements. In *Proceedings of the Workshop on Management of Replicated Data*, pages 133–136, 1990.

[SRK92] Sheth A., M. Rusinkiewicz and G. Karabatis. Using Polytransactions to Manage Interdependent Data. In Elmagarmid, A. K. (Ed.), *Database Transaction Models for Advanced Applications*, pages 555–582, Morgan-Kaufmann, 1992.

[SKS91] Soparkar N., H. Korth and A. Silberschatz. Failure–Resilient Transaction Management in Multidatabases. *IEEE Computer*, Vol. 24, No. 12, pages 28–36, 1991.

[VE91] Veijalaine J., F. Eliassen and Holtkamp. The S–transaction Model. In Elmagarmid A. K. (Ed.), *Database Transaction Models for Advanced Applications*, pages 467–514, Morgan Kaufmann, 1992.

[WR92] Wächter H. and A. Reuter. The ConTract Model. In Elmagarmid A. K. (Ed.), *Database Transaction Models for Advanced Applications*, pages 219–264, Morgan Kaufmann, 1992.

[WQ87] Wiederhold G. and X. Qian. Modeling Asynchrony in Distributed Databases. In *Proceedings of the 3rd International Conference on Data Engineering*, pages 246–250, 1987.

[WV90] Wolski A. and J. Veijalainen. 2PC Agent Method: Achieving Serializability in Presence of Failure Source in a Heterogeneous Multidatabase. In *Proceedings of PARBASE-90 Conference*, pages 321-330, 1990.

[WYP92] Wu K.L., P.S. Yu and C. Pu. Divergence Control for Epsilon-serializability. In *Proceedings of the 8th International Conference on Data Engineering*, 1992.

[ZE93] Zhang A. and A. Elmagarmid. On global Transaction Scheduling in Multidatabase Systems. In *Proc. of the Second International Conference on Parallel and Distributed Information Systems*, pages 117–124, 1993.

[ZE93] Zhang A. and A. Elmagarmid. A Theory of Global Concurrency Control in Multidatabase Systems. *VLDB Journal*, 2(3), pages 331–360, 1993.