TAILOR-MADE CONCURRENCY CONTROL - DISTRIBUTED TRANSACTIONS AS A CASE

Prof. Dr. Techn. Mads Nygard
The Database Technology Section, Sintef Delab
The Norwegian Institute of Technology, Trondheim Norway

ABSTRACT
This paper applies a model for distributed databases and transactions with a distinction between global and local correctness criteria. The global requirements per system are weaker than the local requirements per site. The paper presents an application which suits such a two-level division. The main motivation for our investigation is based on the fact that the commonly used correctness criteria for concurrency control and recovery, serializability and total recoverability, are very strict criteria. The use of more relaxed criteria (allowing more true parallel behaviour and more true partial behaviour) is therefore very appealing - as long as this can be achieved without compromising safety or applicability. The main paradigm in our approach is based on the observation that relatively little knowledge about the databases and transactions can lead to major gains in system throughput. This allows specific systems to have more tailor-made correctness criteria.

We analyse a specialized type of distributed database, the skeleton-database, and a specialized type of distributed transaction, the wander-transaction. Wander-transactions accessing a skeleton-database allow breaks with both the common serializability criterion and the common total recoverability criterion. Our main emphasis here is on the nonserializability aspect. The primary goal of this work is to designate correctness criteria for controlling local and global parallelism. The secondary goal is to specify priority rules for handling local and global criteria breaks. Wander-transactions accessing a skeleton-database experience dynamic priorities. Our resulting concept, priority serializability, gives increased parallelism without compromising safety.

INTRODUCTION

Background:
For many distributed databases the type of global integrity constraints implied by the databases collectively is under discussion. Any such constraints come in addition to the local integrity constraints implied by the databases separately. By integrity constraints we mean any associated rules which couple the given database items in some way. Likewise, for many distributed transactions the type of global semantic requirements induced by the transactions per system is under discussion. Any such requirements come in addition to the local semantic requirements induced by the transactions per site. By semantic requirements we mean any overall information describing the assumed transaction results in some way. See for example Elmagarmid (1992), Breitbart (1992) and Bhargouti (1991). It is a fact that in some distributed databases global integrity constraints do exist but are weaker than the local integrity constraints. It is also a fact that in some distributed transactions global semantic requirements do exist but are weaker than the local semantic requirements. This opens up for nonserializability globally and partial recoverability globally - on top of serializability locally and total recoverability locally.

Context:
A knowledge of the concurrency control material in Bernstein (1987) and Papadimitriou (1986) will benefit the reader but it is not a prerequisite.

Contents:
The “Alternative Database And Transaction Types” chapter presents the skeleton-database and wander-transaction concepts - and investigates the new notions from a general viewpoint. The “Relaxed Correctness Criteria” chapter then designates the correctness criteria for concurrency control and specifies the priority rules for criteria break handling, while the “Mechanisms And Examples” chapter investigates corresponding mechanisms and examples.
The skeleton-database represents sets of close substitutes for diverse articles. Effectively we investigate an organized coupling of cooperating databases. Each specific database contains a complementing set of article-offers - and the different databases contain substituting sets for these article-offers. Further, the databases may be accessed simultaneously in the sense that article-offers in one database may be "held" while article-offers in another database are checked - and the accesses to all participating databases in a skeleton-database will be collectively controlled. Our approach allows nonserializability and partial recoverability. We arrive at multi-level correctness criteria with respect to both consistency preservation and atomicity assurance.
The wander-transaction traverses a skeleton-database trying to seize an optimal set - complying with any combined conditions on the attribute-values - of specific articles. Actually we discuss a traversing race among
competing transactions. A transaction may require diverse articles of several types - and several articles of a single type. And a transaction/buyer may acquire articles directly - i.e. it corresponds to an on-line read-and-write service and not only an off-line read-only service. Further, a buyer may consider more static attributes like quality and price - as well as more dynamic attributes like no-left and who-purchased. Our concept allows transactions to build up priority during their traversing race. Acquiring an article-offer early may increase the priority of the corresponding buyer - while checking an article-offer later may decrease the priority of the corresponding buyer.

**ALTERNATIVE DATABASE AND TRANSACTION TYPES**

**Skeleton-Databases**

Imagine a collection of separately run department stores selling similar types of articles. The current available qualities, prevailing prices, remaining quantities and confirmed reservations of articles naturally vary among the department stores. Their article information will have to exist in \( n \) variants and may have to be stored at \( n \) sites. A typical database corresponding to three department stores offering three specific articles is illustrated in Fig. 1.

Here \( x, y \) and \( z \) signify different occurrences of sites, while \( a, b \) and \( c \) signify different types of items. So \( y_a, y_b \) and \( y_c \) represent different item-types available as article-kinds at a unique site or department store. We talk about a complementing set of article-offers. Further \( x_a, y_a \) and \( z_a \) represent a unique item-type available as article-variants at different sites or department stores. We talk about a substituting set of article-offers.

Consider the article-information in the previous figure as a distributed database consisting of a set of local databases. Each set of items connected with horizontal lines corresponds to the intra-base view, while the set of items encircled by a vertical oval corresponds to an inter-base view. The different local databases of this non-replicated global database are not interrelated by normal integrity constraints. But within each local database there may be any kind of integrity constraints relating the local items in some way. The general picture will be as in Fig. 2.
We are actually specifying an alternative database type where some integrity constraints (i.e. those between the separate local databases in a global database) are completely missing. Hence we have complete knowledge about a well-defined part of the integrity constraints. The non-existence of integrity constraints between clearly identifiable database parts is easy to represent and manipulate. We choose to call a global database of this type a skeleton-database.

In a skeleton-database there are no inter-site integrity constraints. This means that the substituting set of article-offers of for example type B (i.e. x_b, y_b, and z_b) has no attached rules restricting the sets of possible attribute-values that may coexist. But there may be some intra-site integrity constraints. This means that the complementing set of article-offers at for example site Y (i.e. y_a, y_b, and y_c) may have some attached rules restricting the sets of possible attribute-values that may coexist.

Examples of such integrity constraints are:

\[
\text{value(attribute}_{i}(y_b)) \geq \text{value(attribute}_{i}(y_c))
\]

\[
\text{value(attribute}_{i}(y_a)) + \text{value(attribute}_{j}(y_c)) \leq \text{value(attribute}_{i}(y_b))
\]

Note that a typical database item may have more attributes than the four we have introduced.

**Wander-Transactions**

Imagine a simple but interesting operation requiring the cheapest article-variant of type B. The specification of the corresponding transaction will be:

\[ T_1: \text{Acquire One}_B: \text{Lowest-Price} \]

The meaning of such an access-predicate is that the corresponding transaction should, after first checking (i.e. retrieving) some or all of the available article-offers, then acquire (i.e. update) the one article-offer fulfilling the given condition in the best way. This is illustrated in Fig. 3. Here the operation checks the current offers by retrieving all existing variants of the specified article-kind, finds one cheapest offer by comparing the prices and consulting the quantities, and acquires the best offer with at least one instance left by updating this variant of the article-kind.
To check an offer means to retrieve its quality, price, no-left (i.e. quantity) and who-purchased (i.e. reservation-list) attributes. A reservation-list reflects the sequence of reservations already made. To acquire an offer means to decrement the quantity (> 1) by one and add the buyer-identity to the reservation-list. The who-purchased attribute will produce a delivery-list for the department store, so the sequence of buyer-identities is important. The execution result with the article-offer at site Y designated as best will be:

\[ T_1: R_i(x_b) R_i(y_b) R_i(z_b) W_i(y_b) \]

We are actually specifying an alternative transaction type where the relationship between the read- and write-sets of the transactions is known (i.e. a reads-before-write case) - and where the functional objective of the transactions is fixed (i.e. to acquire articles). Hence, we have both some high-level information about the syntax of the transactions and some overall information about the semantics of the transactions. We are particularly concentrating on a specific type of transaction. We choose to call a transaction of this type a wander-transaction.

In a wander-transaction there are no inter-site value-dependencies. This means that values read among several sites are only used to decide which subset of items to update - and will not be used to calculate which values to write to this subset of items. But there may be some intra-site value-dependencies. This means that values read within a single site may have to be applied to calculate which values to write to its designated subset of items.

For example, assume that all the nine database items in Fig. 1 in the “Skeleton-Databases” section are checked, and assume that \( y_b \) is both the only database item acquired of type B and the only database item acquired at site Y. Then the values read from the substituting set of article-offers of type B (i.e. \( x_b, y_b \) and \( z_b \)) are only used to decide that \( y_b \) is to be updated. In contrast, the values read from the complementing set of article-offers at site Y (i.e. \( y_a, y_b \) and \( y_c \)) have to be applied to calculate which value to be written to this \( y_b \).

**Initial Illustrations**

Now we only consider the simplest case - i.e. with one item per site. In general there are several items per site, and the different sites must not necessarily have entries for exactly corresponding sets of items.

Let us imagine a skeleton-database containing the following three item-variants:

<table>
<thead>
<tr>
<th></th>
<th>X:</th>
<th>Y:</th>
<th>Z:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality</td>
<td>85%</td>
<td>90%</td>
<td>95%</td>
</tr>
<tr>
<td>Price</td>
<td>5 $</td>
<td>10 $</td>
<td>15 $</td>
</tr>
<tr>
<td>No-Left</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>Who-Pur.</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

The item-variants represent the one and only item-type occurring at the three different sites. Or to put it another way: each specific quadruple corresponds to a specific department store’s offer of the single article-kind.

We look at the following two wander-transactions:

- \( T_1: \) Acquire One: Lowest-Price \& Quality ≥ 85 %
- \( + \) Acquire One: Lowest-Price \& Quality ≥ 80 %

- \( T_2: \) Acquire One: Lowest-Price \& Quality ≥ 75 %

Recall that a buyer may be interested in more than one article-instance - as \( T_1 \) is here. The item-variants that the two wander-transactions will acquire if executed in isolation are \( T_1: x + y \) and \( T_2: x \).

A possible schedule corresponding to the two wander-transactions being executed in parallel is:

\[ H_1 = T_1: R_i(x) W_i(x) \]
\[ T_2: R_i(y) R_i(z) W_i(z) \]

The description of a schedule is split among the involved transactions. Thus, the horizontal axis represents “time”, while the vertical axis represents “space”. That the wander-transactions access the item-variants in different sequences is not important. It has only been employed for simplicity reasons. One reason for not
checking all article-offers before acquiring an article (as $T_1$ does with $x$ here) is that a very good offer should be
acted upon as soon as possible. Note the possible race-situation among wander-transactions for a favourable
offer.\(^6\)

The results stem from the facts that initially there is only one instance left of both item-variant $x$ and item-variant $y$. When $T_1$ would like to acquire $x$, there is none left and it has to settle for $y$. Further, when $T_1$ would
like to acquire $y$, there is none left and it has to settle for $z$. The item-variants that the two wander-transactions
will acquire if executed in the $H_I$-schedule are $T_1: x + z$ and $T_2: y$.

Observe that these are not the same as in the isolated cases. In the “Initial Evaluations” section we shall see that
the $H_I$-schedule will not be allowed by the designated correctness criteria. Typical examples of schedules that
will be allowed by these correctness criteria shall be shown in the “Concurrency Control Examples” section.

Inherent Freedom

We will consider some consequences of defining and using skeleton-databases and wander-transactions. We will
specifically look at the possible needs for serializability and total recoverability.

The normal correctness criterion for concurrency control corresponds to the one-after-the-other effect. This is
illustrated in Fig. 4. Such serializability means that the effect of handling several transactions being run in
parallel should be as if they were executed in some, unknown, serial order. This is a very limiting criterion. Its
necessary strength stems from the difficulty in knowing, representing and manipulating both general integrity
constraints of databases and general semantic requirements of transactions.

The lack of some integrity constraints in a skeleton-database leads to a certain degree-of-freedom in
synchronizing accesses to such databases. The existence of some semantic information about a wander-
transaction then fixes a particular point in this solution span for synchronization of such transactions. Consider
for instance using locking as a concurrency control mechanism. Is 2PhaseLock synchronization then necessary
for wander-transactions accessing a skeleton-database? The answer is no! It is quite adequate to always hold the
best offer currently found and constantly free already obsolete offers. Holding corresponds to locking and
freeing to unlocking. Carrying out this principle recursively means that eventually an article is locked after
another is unlocked. Hence 2PhaseLocking is violated.

\[\begin{array}{c}
T_1 \\
R_1 \{ W_1, R_1 \} \\
T_2 \\
R_2 \{ W_2, R_2 \} \\
T_n \\
R_n \{ W_n, R_n \}
\end{array} \rightarrow \begin{array}{c}
T_i \\
R_i \{ W_i \} \\
T_j \\
R_j \{ W_j \} \\
T_k \\
R_k \{ W_k \}
\end{array} \]

Figure 7. The One-After-The-Other effect of transactions execution

Relaxed concurrency control has been used in for instance Garcia-Molina (1982) (allowing special
arrangements for read-only transactions in centralized systems) and Du (1989) (allowing special arrangements
for local-only transactions in distributed systems). Our aim is to arrive at relaxed concurrency control for all
wander-transactions. Effectively, true parallel behaviour may still give sensible results. This will be the main
theme of the “Relaxed Correctness Criteria” and “Mechanisms And Examples” chapters.\(^7\)

The normal correctness criterion for recovery corresponds to the all-or-nothing effect. This is illustrated in Fig.
5. Such total recoverability means that the effect of dealing with a transaction being interrupted in the middle
should be as if it was finished totally - or as if it was not started at all. This is again a very limiting criterion. Its

\(^6\) Hence “Lowest-Price” or “Highest-Quality” is not always to be interpreted literally - as might be
indicated in the “Wander-Transactions” section. See also later examples.

\(^7\) Note that we require normal serializability between the group of “selling transactions” and the group of
“buying transactions” - and within the group of “selling transactions”. It is within the group of “buying
transactions” that we may allow some non-serializability.
necessary strength once more stems from the difficulty in knowing, representing and manipulating both general integrity constraints of databases and general semantic requirements of transactions.

![Figure 8. The All-Or-Nothing effect of transaction execution](image)

Wander-transactions accessing a skeleton-database may apply partial commitment instead of total commitment. This means that at particular points during its execution a specific transaction may commit its hitherto purchased items - instead of having to commit all its purchased items together at the end. Each purchase corresponds to a single write, so committing a set of purchases corresponds to committing a set of writes. A reason for applying partial commitment instead of total commitment is that at particular points a specific transaction may be able to live with the items acquired by then - without necessarily having to get hold of the remaining required items. Allowing such step-wise committing transactions does not mean that corresponding subtransactions become totally independent. Actually these will be priority-connected: early writes in a specific transaction build up priority for later writes in the same transaction. Further, early commits secure purchases from being aborted if the transaction waits too long.

Partial commitment has been applied with nested transactions, and this context has been analysed in Bancilhon (1985), Beeri (1986) and Korth (1988). Our point is not that partial commitment is to be used but rather to discuss how it should be controlled. Effectively, even true partial behaviour may still give sensible results. Nygård (1994b) has covered this.

**RELAXED CORRECTNESS CRITERIA**

**Basic Concurrency Control**

We must first look at the transaction conflict patterns that may occur corresponding to two transactions accessing a single item-variant. Here wr, rw or ww indicates a write-read, read-write or write-write conflict respectively - see Bernstein (1987) or Papadimitriou (1986). One or more "&"s couple a set of conflicts that have to occur together in a specific schedule. In contrast, one or more "/"s separate sets of conflicts that have to occur alternatively in a specific schedule. Further, a conflict marked with an "*" contributes in the reverse way of a non-marked conflict. Hence, if a non-marked conflict contributes $T_1 \rightarrow T_2$, then a *-marked conflict contributes $T_2 \rightarrow T_1$. Each separate alternative is accompanied by a corresponding example schedule.

The set of possible results is as follows for the reads-before-write case:

**Both read-only** $\Rightarrow$ *none*

$$
H_a =
\begin{align*}
T_1 : & R_1(y) \\
T_2 : & R_2(y)
\end{align*}
$$

**One reads-only, one reads-and-writes** $\Rightarrow$ *rw / wr*

$$
H_b =
\begin{align*}
T_1 : & R_1(y) \\
T_2 : & R_2(y)W_2(y)
\end{align*}
$$

$$
H_c =
\begin{align*}
T_1 : & R_1(y)W_1(y) \\
T_2 : & R_2(y)
\end{align*}
$$
Both read-and-write ⇒ wr & rw & ww / rw & rw* & ww*

\[ H_d = \]
\[ T_1: R_1(y)W_1(y) \]
\[ T_2: R_2(y)W_2(y) \]

\[ H_r^* = \]
\[ T_1: R_1(y) W_1(y) \]
\[ T_2: R_2(y) W_2(y) \]

The alternative with *-marked conflicts exactly represents the possible lost update situations. The corresponding schedule has been marked with a *-symbol.

• With respect to the synchronization of typical wander-transactions, which of these transaction conflict patterns should we pay attention to and which should we not? An asymmetric assumption is that whenever a transaction acquires an item-variant, the corresponding article-offer can become only less attractive and never more attractive to later transactions. This means that the lower the current quantity and the longer the current reservation-list, the less interesting the article-offer appears. For simplicity reasons we have also anticipated that each wander-transaction only needs one instance of an article. In the following, a reads-and-writes block being changed into a reads-only action further means that another variant of the corresponding item will/must be acquired instead.

• The rw & rw* & ww* alternative embedded in the He*-schedule represents a lost update situation. To disallow such a pattern guarantees that a transaction which, after having checked an article through a read, decides to acquire the article through a write, actually gets an instance of this article. In contrast, to allow such a pattern does not guarantee this property. This pattern is termed \( a_p \) for parallel acquiring.

Actually, a tentative serialization of the corresponding transactions has two typical results:

\[ R_2(y) R_1(y) W_2(y) W_1(y) \Rightarrow R_1(y) W_1(y) R_2(y) W_2(y) \]
\[ R_2(y) R_1(y) W_2(y) W_1(y) \Rightarrow R_1(y) W_1(y) R_2(y) \]

The first outcome occurs when there are article-instances enough for both buyers, and the second buyer does not mind that the first buyer gets the article before him. In contrast, the second outcome occurs if there are not article-instances enough for both buyers - or the second buyer does mind that the first buyer gets the article before him.

• The wr & rw & ww alternative embedded in the H\(_d\)*-schedule reflects a situation where two transactions in turn check and decide each to acquire an instance of a specific article. To take notice of such a pattern means that the sequence in which instances of a specific article are acquired does matter. In contrast, not to take notice of such a pattern means that the sequence does not matter. This pattern is termed \( a_s \) for sequential acquiring.

Actually, a tentative reversal of the corresponding transactions has two typical results:

\[ R_1(y) W_1(y) R_2(y) W_2(y) \Rightarrow R_2(y) W_2(y) R_1(y) \]
\[ R_1(y) W_1(y) R_2(y) W_2(y) \Rightarrow R_2(y) W_2(y) R_1(y) \]

The first outcome occurs when the first buyer does not mind that the second buyer gets the article before him. In contrast, the second outcome occurs if the first buyer does mind that the second buyer gets the article before him.

• The wr alternative embedded in the H\(_d\)*-schedule indicates that a transaction, after having checked an article through a read, decides not to acquire the article. This may be because the article is not the optimal choice - or because the conflicting transaction has acquired the last instance of it through a write. The last possibility makes it important to pay attention to such patterns. If the transaction conflict had been reversed, the original transaction could have acquired the last instance of the article instead. This pattern is termed \( c_a \) for checking after.

September 1995 59
Actually, a tentative reversal of the corresponding transactions has three typical results:

\[
\begin{align*}
R_i(y) & W_i(y) R_2(y) \Rightarrow R_2(y) R_i(y) W_i(y) \\
R_i(y) & W_i(y) R_2(y) \Rightarrow R_3(y) W_2(y) R_i(y) W_i(y) \\
R_i(y) & W_i(y) R_2(y) \Rightarrow R_2(y) W_2(y) R_i(y)
\end{align*}
\]

The first outcome occurs when the second buyer’s negative decision is not related to the first buyer’s positive decision. Further, the second outcome occurs when/ if only the second buyer does mind that the first buyer gets the article before him, and not also vice versa. In contrast, the third outcome occurs if there are not article-instances enough for both buyers - or each buyer does mind that the other buyer gets the article before him.

- **The rw alternative** embedded in the \( H_b \)-schedule again indicates that a transaction, after having checked an article through a read, decides not to acquire the article. This may only be because the article is not the optimal choice - as there is still at least one instance left. This makes it unimportant to pay attention to such patterns. If the transaction conflict had been reversed, the results would have been the same for both transactions anyway. This pattern is termed \( c_b \) - for checking before.

Actually, a tentative reversal of the corresponding transactions has a single typical result:

\[
\begin{align*}
R_i(y) & R_2(y) W_2(y) \Rightarrow R_2(y) W_2(y) R_i(y)
\end{align*}
\]

The single outcome occurs as the first buyer’s negative decision cannot be related to the second buyer’s positive decision.

Thus, the \( c_b \)-patterns should not be included in the synchronization, while the \( a_c \)-patterns should. Further, it is most natural to disallow an \( a_c \)-pattern - i.e. to have it forced into either an \( a_c \)- or a \( c_b \)-pattern. Also, it is quite natural to take notice of an \( a_c \)-pattern - i.e. to let the acquiring-sequence of article-instances matter.\(^8\) Note that the \( c_b \)- and \( a_c \)-patterns will be included in the synchronization by paying attention to all occurring write-read conflicts - i.e. irrespectively of whether each is accompanied by corresponding read-write and write-write conflicts or not. Hence, we will require full local serializability (per item - or per site) but allow some global non-serializability (per system).

We can now illustrate these statements with reference to Fig. 1 in the “Skeleton-Databases” section. Basically we consider having only one item per site.

In the following schedule reflecting a **double check-after situation**, \( H_{DCA} = \)

\[
\begin{align*}
T_1: & R_i(x)W_i(x) \quad R_i(y) ... \\
T_2: & R_2(x)R_2(y)W_2(y) \quad ...
\end{align*}
\]

there are two \( c_b \)-patterns embedded contributing in reverse ways. The x-accesses contribute \( T_1 \rightarrow T_2 \), and the y-accesses contribute \( T_2 \rightarrow T_1 \). As indicated above, we can neither reverse the x-accesses nor reverse the y-accesses without possibly changing the decisions. Hence, there is no way to obtain an “equivalent” serial schedule here, and we cannot allow such a schedule.

In the following schedule reflecting an **inconsistent retrieval situation**, \( H_{IRI} = \)

\[
\begin{align*}
T_1: & R_i(x)W_i(x) \quad R_i(y)W_i(y) ... \\
T_2: & R_2(x)R_2(y) \quad ...
\end{align*}
\]

there is one \( c_c \)-pattern and one \( c_b \)-pattern embedded contributing in reverse ways. The x-accesses (corresponding to the \( c_c \)-pattern) contribute \( T_1 \rightarrow T_2 \), and the y-accesses (corresponding to the \( c_b \)-pattern) contribute \( T_2 \rightarrow T_1 \). As indicated above, we cannot reverse the x-accesses but we can reverse the y-accesses without possibly changing the decisions. Hence, here there is a way to obtain an “equivalent” serial schedule, \( T_1/T_2 \), and we can allow such a schedule.

---

8. Its importance may be based on the specific buyer-identities in *who-purchased* - or only on the number of buyer-entries in *who-purchased*.
In the following schedule reflecting another inconsistent retrieval situation,

\[ H_{R1} = \]
\[ T_1: R_1(x) \]
\[ T_2: R_2(x) W_2(x) R_2(y) W_2(y) \]

there is one \( c_b \)-pattern and one \( c_a \)-pattern embedded contributing in reverse ways. The \( x \)-accesses (corresponding to the \( c_b \)-pattern) contribute \( T_1 \to T_2 \), and the \( y \)-accesses (corresponding to the \( c_a \)-pattern) contribute \( T_2 \to T_1 \). As indicated above, we can reverse the \( x \)-accesses but we cannot reverse the \( y \)-accesses without possibly changing the decisions. Hence, here there is a way to obtain an “equivalent” serial schedule, \( T_2 T_1 \), and we can allow such a schedule.

In the following schedule reflecting a double check-before situation,

\[ H_{DCB} = \]
\[ T_1: R_1(x) W_1(x) \]
\[ T_2: R_2(x) R_2(y) W_2(y) \]

there are two \( c_b \)-patterns embedded contributing in reverse ways. The \( x \)-accesses contribute \( T_1 \to T_2 \), and the \( y \)-accesses contribute \( T_2 \to T_1 \). As indicated above, we can reverse either the \( x \)-accesses or the \( y \)-accesses without possibly changing the decisions. Hence, here there are two ways to obtain an “equivalent” serial schedule, \( T_2 T_1 \) or \( T_1 T_2 \), and we can allow such a schedule.

Note that if \( x \) and \( y \) had belonged to a unique site instead of to two different sites, the situations reflected in both \( H_{R1}/H_{R2} \) and \( H_{DCB} \) could not have been allowed in general. This difference stems from the lack of inter-site integrity constraints but existence of intra-site integrity constraints. The ban on intra-site inconsistent retrieval and double check-before situations is the local generalization of the ban on single-item lost update situations. (Both inter-site and intra-site double check-after situations are also prohibited).

**Formal Correctness Criteria**

Based on our initial discussion in the “Basic Concurrency Control” section we get:

- **In the special cases** where we have only one item per site - or where we have several items per site but no intra-site integrity constraints, we require local serializability per item plus a global condition of non-serializability type.
- **In the general case** where we have several items per site and some intra-site integrity constraints, we require local serializability per site plus the global condition of non-serializability type.

In any case, our 2-level total requirement corresponds to the following class of schedules:

\[ C_{WR} = C_{G: WR} \cap C_{L: WR-RW-WW} \]

To explain these designated criteria we have to introduce some extra notions.

The binary relation \( WR_D(H) \) contains the set of ordered pairs of transactions corresponding to the write-read conflicts in the schedule \( H \) related to items in the whole global database \( D \). \( WR_D(H)^* \) represents the transitive closure of \( WR_D(H) \). Further, the binary relations \( WR-RW-WW_Y(H) \) and \( WR-RW-WW_y(H) \) contain the sets of ordered pairs of transactions corresponding to the write-read, read-write and write-write conflicts in the schedule \( H \) related to items at the local site \( Y \) and the single local item \( y \) respectively. \( WR-RW-WW_Y(H)^* \) and \( WR-RW-WW_y(H)^* \) represent the transitive closures of \( WR-RW-WW_Y(H) \) and \( WR-RW-WW_y(H) \) respectively.

For a schedule \( H \) to be a member of class \( C_{G: WR} \) the binary relation \( WR_D(H)^* \) has to be a partial order. Further, for a schedule \( H \) to be a member of class \( C_{L: WR-RW-WW} \) all the binary relations \( WR-RW-WW_Y(H)^* \) or \( WR-RW-WW_y(H)^* \) (corresponding to each particular site or each single item) have to be partial orders. Hence, for a schedule \( H \) to be a member of our designated class of schedules \( C_{WR} \) it has to be a member of both \( C_{G: WR} \) and \( C_{L: WR-RW-WW} \), where \( L \) for local means either \( S \) for per site or \( I \) for per item. (\( G \) for global means per system).

A schedule being a member of the \( C_{WR} \)-class has one serialization which complies with the write-read conflicts for the global system - plus \( N \) serializations each of which complies with the write-read, read-write and write-
write conflicts for one of the N local subsystems. All these piecewise serializations\(^9\) can be separately pursued, and the 1+N resulting serial schedules\(^{10}\) may thus all be inconsistent with respect to their sequencing - see the example schedules in the "Concurrency Control Examples" section.

The basic lack of RR in the local part of our class specification means that only the following reversal type

\[
R_i(y)...R_j(y) \Rightarrow R_j(y)...R_i(y)
\]

is allowed in the pursuit of equivalent serial schedules for the different local subsystems. The extra lack of RW in the global part of our class specification means that also the following reversal type

\[
R_i(y)...W_j(y) \Rightarrow W_j(y)...R_i(y)
\]

is allowed in the pursuit of an equivalent serial schedule for the single global system.

That the \(c_a\)-patterns should be included in the synchronization while the \(c_b\)-patterns should not, is reflected in the global part. Further, that an \(a_p\)-pattern is not allowed, is reflected in the local part. Effectively, while we do not allow a transaction's basis with respect to a local decision to be invalidated by one or more other transactions - e.g. by not allowing any lost update situations, we do allow a transaction's bases with respect to a global evaluation to be influenced by one or more other transactions - e.g. by allowing some inconsistent retrieval situations.

The resulting effects of our combined local and global requirements may be termed

**Final-State Consistency**

+ 

**Final-Choice Serializability.**

The "Final-State Consistency" effect stems from the local requirement's ability to preserve the integrity constraints in the local databases, and the "Final-Choice Serializability" effect stems from the global requirement's ability to observe the overall goals of the global transactions. The basic goal of wander-transactions is to acquire articles. It is also anticipated that if a first transaction manages to acquire a specific article before a second transaction manages to check the same offer, the first transaction should not have to check another offer after the second transaction has already acquired that article. Hence, the reads-from relation from Bernstein (1987) and Papadimitriou (1986) has to be maintained as a partial order, and the final choices of the wander-transactions will thus be serializable. This means that there will exist a corresponding serial schedule in which their final choices will be the same.

We will achieve effects that may be considered a variant of so-called final-state serializability in the special cases where we have only one item per site - or where we have several items per site but no intra-site integrity constraints. This takes into account the lack of inter-site value-dependencies. In contrast, we will not even achieve this variant of final-state serializability in the general case where we have several items per site and some intra-site integrity constraints. This is due to the existence of intra-site value-dependencies. For a discussion of normal final-state serializability, see Papadimitriou (1986).

To exemplify these statements, consider the following schedule:

\[
H_{WFSS} = \\
T_1: R_1(x_a)W_1(x_a) \quad R_1(y_b)W_1(y_b) \quad ... \\
T_2: R_2(x_a)R_2(y_b) \quad R_2(y_c)W_2(y_c) \quad ...
\]

---

9. Piecewise serialization means only paying attention to the transaction conflicts and the region indicated by the name of the corresponding binary relation.

10. Note that any topological sort of the acyclic directed graph corresponding to a specific partial order gives a serial schedule which is a piecewise serialization of the original schedule.
Assume also that the first of the integrity constraints indicated in the "Skeleton-Databases" section applies. The \( H/vts\)-schedule (which is a variant of the \( H/wi\)-schedule from the "Basic Concurrency Control" section) will be allowed by our concurrency rules, and the topological ordering of \( WR_D^*\) gives:

\[ T_1/T_2 \]

But the value that \( T_2 \) would write to \( y_c \) in this serial schedule might differ from what it will write to \( y_c \) in the \( H/vts\)-schedule as it retrieves \( y_b \) there before \( y_b \) is updated by \( T_1 \). Hence the mentioned variant of final-state serializability is not observed.

**Extra Priority Rules**

Wander-transactions accessing a skeleton-database correspond to a reads-before-write case. This means that a write on a specific item by a specific transaction will always be preceded by a read on the same item by the same transaction.

As in the "Formal Correctness Criteria" section the binary relations \( WR_D(H) \) and \( RW_D(H) \) contain the sets of ordered pairs of transactions corresponding to the write-read conflicts and the read-write conflicts respectively in the schedule \( H \) related to items in the whole global database \( D \). Further, \( WR_D^*(H) \) and \( RW_D^*(H) \) represent the transitive closures of \( WR_D(H) \) and \( RW_D(H) \) respectively.

Observe the high dominance of reads over writes and the high dominance of late writes over early writes inherent in the semantics of wander-transactions. This would for instance make a global requirement which corresponds to the binary relation \( RW_D(Any\ H)^* \) being a partial order very limiting with regard to allowing possible race-situations for favourable offers. Likewise, this makes the global requirement which does correspond to the binary relation \( WR_D(Any\ H)^* \) being a partial order also suitable as a priority mechanism in controlling actual race-situations for favourable offers.

A global WR-order (i.e. the sequencing implied by \( WR_D(H)^* \)) may thus be used in solving breaks with the local requirement per item/site.

* When a **local break** is about to occur, the WR-order is consulted. If the involved wander-transactions (two or more) are ordered, any latest ordered wander-transaction among these is selected for abortion. In contrast, if they are not ordered, any of these wander-transactions may be chosen for abortion. This implies that quickly making decisions with respect to items (i.e. acquiring a specific article through a write after having checked the corresponding offer through a read) can give high priority to a wander-transaction with respect to avoiding its selection for abortion. Hence wander-transactions accessing a skeleton-database experience dynamic priorities. These are not based on when transactions are started but on when actions are executed. Further, they are only based on the write operations - and only on specific write operations being followed by corresponding read operations. A write operation may thus give a transaction priority over some particular transactions but not over some other transactions. Such a use of the binary relation \( WR_D(Any\ H)^* \) as a priority mechanism also requires that it has to be maintained as a partial order.

* When a **global break** is about to occur, the wander-transaction whose write is part of the write-read conflict about to close a cycle in the graph corresponding to the global WR-relation is selected for abortion. Hence, among the involved wander-transactions (two or more) the one issuing the latest corresponding write will normally be chosen for abortion. This means that quickly making decisions with respect to items can even give high priority to a wander-transaction with respect to it getting hold of other attractive offers. Maintaining a global WR-order may thus even be used to influence the local WR-RW-WW-order - i.e. the sequencing implied by \( WR-RW-WW^*(H)^* \). The final-choice serializability mentioned in the "Formal Correctness Criteria" section will depend on the dynamic priorities of wander-transactions, and the resulting concept may be termed priority serializability.

So our designated criteria for concurrency control induce direct reactions on rule breaks. This means that when an access operation is requested which would lead to a global or local break an appropriate abortion is invoked immediately. The reasons are that a later access operation cannot fix the break which is about to occur as we employ conflict based concurrency control, and a later abort operation by a transaction itself is not normally to be expected. This corresponds to a pessimistic concurrency control. For discussions of conflict based versus view based concurrency control and pessimistic versus optimistic concurrency control, see Bernstein (1987).

Just as a later abort operation by a transaction itself is not normally to be expected, a later commit operation by a transaction is not to be allowed totally uncontrolled. This important topic is the main theme of Nygård (1994b). Actually, there we have specified a set of reliability rules which may be combined in an orthogonal way. Note that there is a trade-off between reliability (with regard to guaranteeing recovery properties) and
concurrency (with regard to allowing parallel activities). Our reliability rules will be linked to the priority rules with which our designated criteria for concurrency control have been extended. Finally, the result will be n-level recovery criteria in addition to our 2-level concurrency control criterion - requiring full local serializability but allowing some global non-serializability.

Initial Evaluations

Let us again look at the example from the "Initial Illustrations" section. Here we get the following results:

\[ WR - RW - WW_x(H_1) = \{ (T_1, T_2) \} \]
\[ WR - RW - WW_y(H_1) = \{ (T_2, T_1) \} \]
\[ WR - RW - WW_z(H_1) = \{ (T_2, T_1) \} \]
\[ WR_d(H_1) = \{ (T_1, T_2), (T_2, T_1) \} \]

Hence, \( WR-RW-WW_x(H_1)* \), \( WR-RW-WW_y(H_1)* \) and \( WR-RW-WW_z(H_1)* \) are all partial orders, but \( WR_d(H_1)* \) is not a partial order. According to the criteria in the "Formal Correctness Criteria" section we must conclude:

\[ H_1 \not\in C_{wr} \]

Further, the read of x by \( T_2 \) contributing to the \( (T_1, T_2) \)-entry in \( WR_d(H_1) \) occurs before the read of y by \( T_1 \) contributing to the \( (T_2, T_1) \)-entry in the same binary relation. According to the rules in the "Extra Priority Rules" section the intermediate result is Aborting \( T_2 \), and this shall be enforced when \( R_i(y) \) occurs. After later Rescheduling \( T_2 \), the final schedule is:

\[ H_2 = \]
\[ T_1: R_1(x)R_1(y)R_1(z)W_1(y) \]
\[ T_2: R_2(x)R_2(y)R_2(z)W_2(z) \]

An underlying assumption is that when \( R_i(y) \) is requested and thus invokes an abortion of \( T_2 \), \( R_i(y) \) itself is not granted until the rollback of \( T_2 \) is completed. Now we get the following results for this serial schedule:

\[ WR - RW - WW_x(H_2) = \{ (T_1, T_2) \} \]
\[ WR - RW - WW_y(H_2) = \{ (T_1, T_2) \} \]
\[ WR - RW - WW_z(H_2) = \{ (T_1, T_2) \} \]
\[ WR_d(H_2) = \{ (T_1, T_2) \} \]

Thus, \( WR-RW-WW_x(H_2)* \), \( WR-RW-WW_y(H_2)* \), \( WR-RW-WW_z(H_2)* \) and \( WR_d(H_2)* \) are all partial orders. According to the criteria in the "Formal Correctness Criteria" section we may conclude:

\[ H_2 \in C_{wr} \]

The item-variants that the two wander-transactions will acquire as executed in this schedule are \( T_1: x + y \) and \( T_2: z \).

Observe that these are neither the same as in the isolated cases nor the same as in the \( H_1 \)-schedule. The \( H_1 \)-schedule reflects a double check-after situation, see the "Basic Concurrency Control" section.
MECHANISMS AND EXAMPLES

Concurrency Control Mechanisms

We have for our combined local and global criteria:

\[
H \in C_{wr} \iff \\
\forall (Y \text{ is a Site}) [WR - RW - WW_y(Any_H) * \text{ is a Partial Order}]
\]

\[
\text{WR}_D(\text{Any}_H) * \text{ is a Partial Order}
\]

Thus, it would be most natural and direct to employ a concurrency control mechanism based on graph-testing both locally and globally. Recall that for any binary relation there is a corresponding directed graph. That the relation is a partial order corresponds to the graph being acyclic. Further, for an acyclic graph there are one or more corresponding topological sorts, while for a cyclic graph there is no corresponding topological sort. Let us denote the graphs corresponding to the relations WR, WR-RW-WW, and WR-RW-WW_u, PSG, CSG_y and CSG respectively. Note that CSG and CSG_y are abbreviations for global and local conflict serialization graph respectively, while PSG is an abbreviation for priority serialization graph. These graphs have been integrated in the later illustrations. Each arc has labels of type \(a^c\) indicating a \(bc\)-conflict on item \(a\). The dashed and non-dashed arcs together constitute the full CSG. The non-dashed arcs alone constitute the PSG part. Each CSG_y part, \(Y\) is a Site, appears as the projection of the full CSG on all \(a^y\)-arcs where \(a \in Y\).

Let us once more look at the example from the “Initial Illustrations” section. From Fig. 6 we have the following topological sorts:

- **CSG**: None
- **PSG**: None

\[
\begin{align*}
\text{CSG}_x : TT & \quad \text{CSG}_y : TT & \quad \text{CSG}_z : TT
\end{align*}
\]

Thus we must conclude:

\[
(H \not\in \text{CSR}) \text{ and } (H \not\in C_{wr})
\]

Our figure clearly illustrates that there is no way to obtain an “equivalent” serial schedule here.

![Diagram](image)

Figure 9. PSG(\(H_i\)) vs. CSG(\(H_i\))

From Fig. 7 we have the following topological sorts:

- **CSG**: \(TT\)  \(TT\)
- **PSG**: \(TT\)

\[
\begin{align*}
\text{CSG}_x : TT & \quad \text{CSG}_y : TT & \quad \text{CSG}_z : TT
\end{align*}
\]

11. The binary relation WR-RW-WW_u(H) contains the set of ordered pairs of transactions corresponding to the write-read, read-write and write-write conflicts in the schedule H related to items in the whole global database D. See the “Formal Correctness Criteria” and “Extra Priority Rules” sections.
Thus we may conclude:

\[
(H_2 \in \text{CSR}) \quad \text{and} \quad (H_2 \in \text{C}_{WR})
\]

Our figure clearly illustrates that there is no need to obtain another “equivalent” serial schedule here.

These deductions are of course in accordance with the conclusions in the “Initial Evaluations” section. We will also indicate how lock-setting could be used as a concurrency control mechanism.

First, we only consider serializability per item as the local criterion. Before a transaction may check an item, it has to request a read-lock. And if the transaction decides not to acquire a specific item, it may release the read-lock as soon as this fact becomes apparent. This may happen immediately if the current offer is not in accordance with the access-predicate - or if it is worse than an already known offer. Alternately this may not happen until later when a better offer is found. This allows breaks with the normal 2PhaseLock/LockUntilEnd-rules for all items globally.

But if the transaction decides to acquire a specific item, it has to upgrade the read-lock to a write-lock. This leads to observation of the normal 2PhaseLock/LockUntilEnd-rules for each item locally. After the transaction has acquired a specific item, it may release the write-lock. But this must happen at a global locked-point or at global end.

Hence, the items which are both retrieved and updated need 2PhaseLocking/LockUntilEnd among themselves - and “upgrade-locking” within each. Further, the items which are only retrieved have to observe a “chain-locking” technique - i.e. the currently best offer has to be kept locked until any better offer is found. This is a double-restricted version of long write-locks and short read-locks, see Gray (1976). It is much too restrictive compared to the graph-testing mechanism.

This mechanism will naturally disallow the H_r-schedule in the “Initial Illustrations” section and allow the H_2-schedule in the “Initial Evaluations” section.

Second, we even consider serializability per site as the local criterion.

Again, the write-locks have to be held until a global locked-point or until global end. Further, the read-locks not upgraded to write-locks have to observe the “chain-locking” technique - and have to be held until a local locked-point or until local end. This is once more much too restrictive.

The global locked-point occurs when all the locks in the whole system are set, while a local locked-point occurs when all the locks for that site are set. The global end occurs when all the accesses in the whole system are terminated, while a local end occurs when all the accesses at that site are terminated.

Concurrence Control Examples

The H_r-schedule in the “Initial Illustrations” section was an indication of what is not allowed by our concurrency control criteria, while the schedules in this section will be indications of what is allowed by our concurrency control criteria. Hence, the “Initial Illustrations” section focused on the limitations inherent in our specifications, while this section will focus on the freedom inherent in our specifications. Again we consider having only one item per site.
Let us imagine a skeleton-database containing the following two item-variants:

<table>
<thead>
<tr>
<th></th>
<th>x:</th>
<th>y:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quality</td>
<td>95%</td>
<td>90%</td>
</tr>
<tr>
<td>Price</td>
<td>10 $</td>
<td>5 $</td>
</tr>
<tr>
<td>No-Left</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Who-Pur.</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

We look at the following three wander-transactions:

1. **T₁**: Acquire One: Lowest-Price & Quality ≥ 85 %
2. **T₂**: Acquire One: Price ≤ 15 $ & Highest-Quality
3. **T₃**: Acquire One: Price ≤ 15 $ & Highest-Quality + Acquire One: Lowest-Price & Quality ≥ 85 %

A possible schedule corresponding to wander-transactions T₁ and T₂ is:

\[ H₃ = \]
\[ T₁: R₁(x) \quad R₁(y) \quad W₁(y) \]
\[ T₂: \quad R₂(x) \quad R₂(y) \quad W₂(x) \]

This schedule reflects a double check-before situation, see the “Basic Concurrency Control” section. From Fig. 8 we have the following topological sorts:

- **CSG**: None
- **PSG**: \( TT₁T₂ \)
- **CSG** \( x : T₁T₂ \)
- **CSG** \( y : T₂T₁ \)

Thus we may conclude:

\[ (H₃ \notin CSR) \quad \text{but} \quad (H₃ \in C_{WR}) \]

Our figure illustrates that there are two ways to obtain a globally equivalent serial schedule here - i.e. either through y-access reversal or through x-access reversal. Schedule \( H₃ \) may also be allowed by the lock-setting mechanism.

![Diagram of concurrency control](image)

**Figure 11. PSG(\( H₃ \)) vs. CSG(\( H₃ \))**

A possible schedule corresponding to wander-transactions T₁ and T₃ is:

\[ H₄ = \]
\[ T₁: \quad R₁(x) \quad R₁(y)W₁(y) \]
\[ T₃: \quad R₃(x)R₃(y)W₃(x)W₃(y) \]
This schedule reflects one inconsistent retrieval situation, see the “Basic Concurrency Control” section. From Fig. 9 we have the following topological sorts:

\[
\begin{align*}
CSG: & \text{None} & PSG: & T_{3}T_{1} \\
CSG_{x}: & T_{1}T_{3} & CSG_{y}: & T_{3}T_{1}
\end{align*}
\]

Thus we may conclude:

\[
(H_{4} \notin CSR) \text{ but } (H_{4} \in C_{WR})
\]

Our figure illustrates that there is one way to obtain a globally equivalent serial schedule here - i.e. through x-access reversal. For schedule \(H_{4}\) even to be allowed by the lock-setting mechanism the specification of transaction \(T_{1}\) has to be changed into for instance:

\(\text{T}_{1}: \text{Acquire One: Price} \leq 7.5 \, \text{\$ & Highest-Quality}\)

![Diagram](image-url)

**Figure 12.** PSG\((H_{4})\) vs. CSG\((H_{4})\)

A possible schedule corresponding to wander-transactions \(T_{1}, T_{2}\) and \(T_{3}\) is:

\[
H_{5} = \begin{align*}
T_{1}: & \text{R}_{1}(x) & \text{R}_{1}(y)W_{4}(y) \\
T_{2}: & \text{R}_{2}(x) & \text{R}_{2}(y)W_{3}(x)W_{2}(x) \\
T_{3}: & \text{R}_{3}(x) & \text{R}_{3}(y)W_{3}(x)W_{3}(y)
\end{align*}
\]

This schedule reflects two inconsistent retrieval situations and a double check-before situation, see the “Basic Concurrency Control” section. From Fig. 10 we have the following topological sorts:

\[
\begin{align*}
CSG: & \text{None} & PSG: & T_{3}T_{1}T_{2} / T_{3}T_{1}T_{2} \\
CSG_{x}: & T_{1}T_{2}T_{3} & CSG_{y}: & T_{2}T_{1}
\end{align*}
\]

Thus we may conclude:

\[
(H_{5} \notin CSR) \text{ but } (H_{5} \in C_{WR})
\]

Our figure illustrates that there are two ways to obtain a globally equivalent serial schedule here - i.e. through y-access reversal for \(T_{2}\) versus \(T_{1}\) and x-access reversal for \(T_{1}\) versus \(T_{3}\), then through either y-access reversal for \(T_{2}\) versus \(T_{1}\) or x-access reversal for \(T_{1}\) versus \(T_{2}\). But schedule \(H_{5}\) will not be allowed by the lock-setting mechanism - irrespectively of any transaction specification changes.
Once more, if x and y had belonged to a unique site instead of to two different sites, the situations reflected in $H_2$, $H_3$ and $H_5$ could not have been allowed in general. As indicated in the "Basic Concurrency Control" section, this is the local generalization of not allowing the situation reflected in $H^*$ from the same section.

Note that if no write could ever precede a read in any transaction, the global part of the membership requirements for class $C_{WR}$ would be ensured for any schedule without any synchronization. Hence the binary relation $WR(g(\text{Any } H)^*)$ would be a partial order per definition. This would for instance be the effects with our criterion for a two-step model, see Papadimitriou (1986). Both the type of access-predicate$^{12}$ of a wander-transaction and the type of item-distribution$^{13}$ in a skeleton-database may lead to a write-precedes-read situation. Observe that reads-before-write refers to a single item-variant, while write-precedes-read refers to different item-variants.

CONCLUSION

To sum up, knowledge about some missing integrity constraints in a distributed database opens the way for global non-serializability - and knowledge about some existing overall goals of distributed transactions leads the way to the add-on to local serializability.

Wander-transactions accessing a skeleton-database correspond to a reads-before-write case. The specified criteria are

$$C_{WR} = C_{G,WR} \cap C_{L,WR-RW-WW},$$

with the last part stemming from the need to preserve integrity constraints in the local databases - and the first part stemming from the need to observe overall goals of the global transactions.

In Nygård (1993a) we have related and compared such total criteria (i.e. the combination of a global criterion per system and the local criterion per item/site) to other known and proposed criteria. This includes the material in Ibaraki (1987), Garcia-Molina (1988), Korth (1988), Gray (1976), Schlageter (1978), Garcia-Molina (1982) and Du (1989). Our concepts are entirely new.

In Nygård (1994b) we have further elaborated on recovery criteria and mechanisms for wander-transactions accessing a skeleton-database. The corresponding n-level model for partial recoverability in distributed databases is combined with the 2-level model for non-serializability introduced in Nygård (1993a) and applied in this paper. See also Nygård (1993b) and Nygård (1994a).

REFERENCES


---

12. This means the high-level goal-specification. See for example the $T_1$-transaction in the $H_1$-schedule from the "Initial Illustrations" section.

13. This means which types are represented at which sites.


