TAILOR-MADE CONCURRENCY CONTROL - DISTRIBUTED TRANSACTIONS AS A CASE

Prof. Dr.Techn. Mads Nygård

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 non-serializability 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 non-serializability 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 non-serializability 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_b, y_b and z_b represent a unique item-type available as article-variants at different sites or department stores. We talk about a substituting set of article-offers.

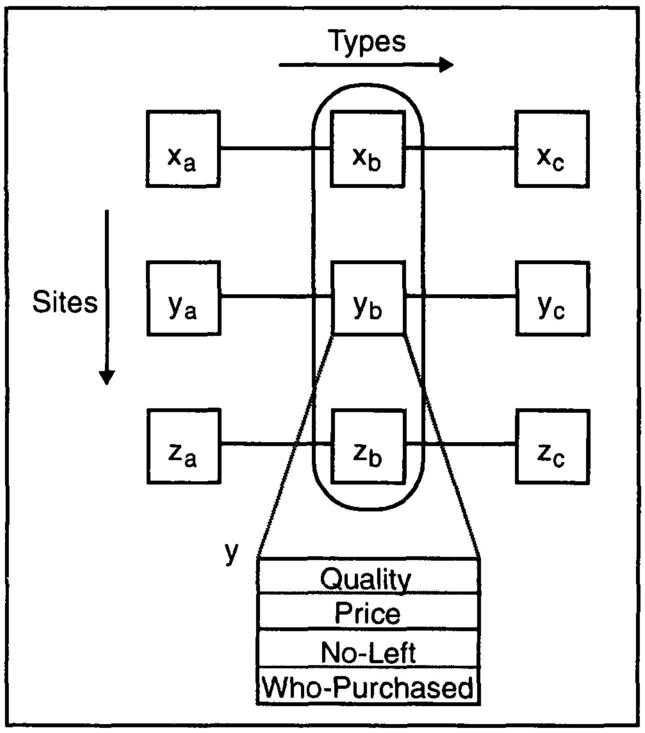

Figure 4. Complementing Sets of Article-Offers at single sites & A Substituting Set of Article-Offers of a specific type

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.

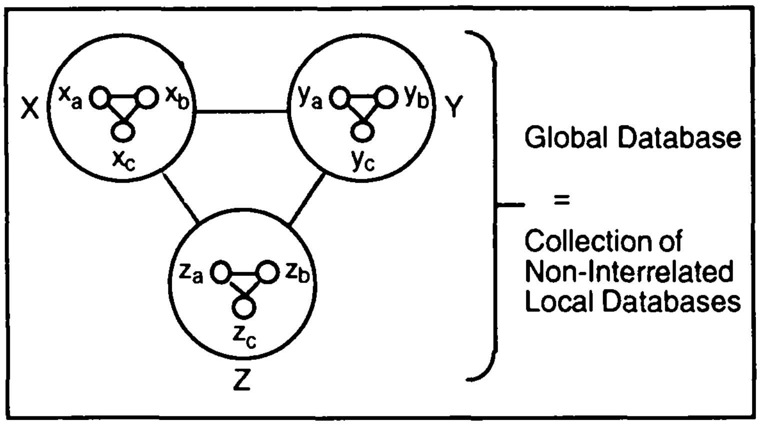

Figure 5. Lacking interrelations in the Alternative Distributed Database

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:

$$\text{value}(\text{attribute}_i(y_b)) \geq \text{value}(\text{attribute}_i(y_c))$$

$$\text{value}(\text{attribute}_j(y_b)) + \text{value}(\text{attribute}_j(y_c)) \leq \text{value}(\text{attribute}_j(y_a))$$

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: Acquire One$_B$: 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.

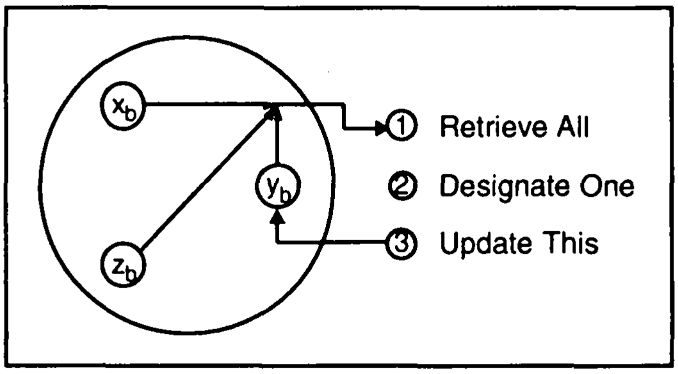

Figure 6. Actions of the Alternative Distributed Transaction

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:

$$T_1\text{: } R_1(x_b)\ R_1(y_b)\ R_1(z_b)\ W_1(y_b)$$

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:

x:	y:	z:
Quality = 85%	Quality = 90%	Quality = 95%
Price = 5 $	Price = 10 $	Price = 15 $
No-Left = 1	No-Left = 1	No-Left = 2
Who-Pur. = -	Who-Pur. = -	Who-Pur. = -

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:

$$\begin{array}{ll} T_1\text{: Acquire One: Lowest-Price \& Quality} \geq 85\ \% \\ \ \ + \text{ Acquire One: Lowest-Price \& Quality} \geq 80\ \% \\ T_2\text{: Acquire One: Lowest-Price \& Quality} \geq 75\ \% \end{array}$$

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:

$$\begin{array}{llll} H_1 = & & & \\ T_1\text{:} & & R_1(x)W_1(x) & & R_1(y)R_1(z)W_1(z) \\ T_2\text{:} & R_2(z)R_2(y) & & R_2(x)W_2(y) & \end{array}$$

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_2 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_1-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_1-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 where 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.

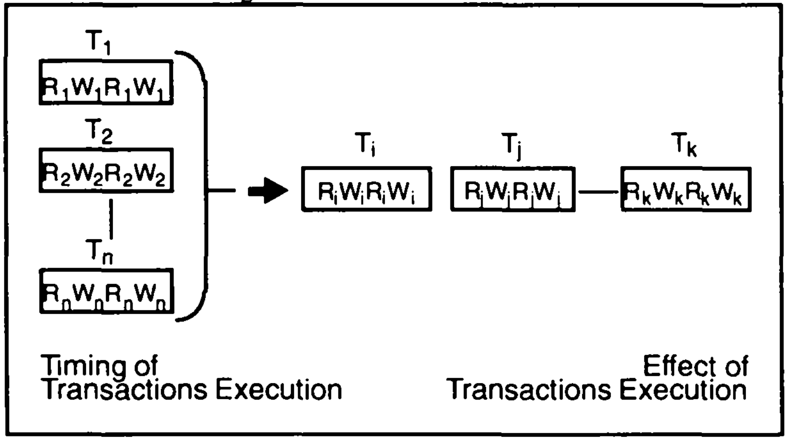
T1
R1W1R1W1
T2
R2W2R2W2
Tn
RnWnRnWn
Ti
RiWiRiWi
Tj
RjWjRjWj
Tk
RkWkRkWk
Timing of Transactions Execution
Effect of Transactions Execution

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.

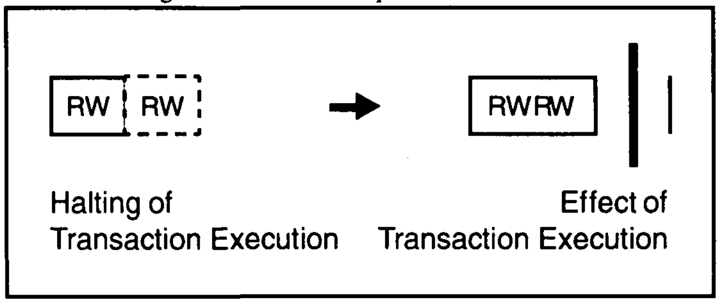
RW
RW
RWRW
Halting of Transaction Execution
Effect of Transaction Execution

Figure 8. The All-Or-Nothing effect of transaction execution

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 ⇒ <none>

H_a =
T_1: $R_1(y)$
T_2: $R_2(y)$

One reads-only, one reads-and-writes ⇒ rw / wr

H_b =
T_1: $R_1(y)$
T_2: $R_2(y)W_2(y)$

H_c =
T_1: $R_1(y)W_1(y)$
T_2: $R_2(y)$

Both read-and-write $\Rightarrow$ wr & rw & ww / rw & rw* & ww*

$$H_d =$$
$$\begin{array}{lll} T_1: & R_1(y)W_1(y) & \\ T_2: & & R_2(y)W_2(y) \end{array}$$

$$H_e^+ =$$
$$\begin{array}{lllll} T_1: & & R_1(y) & & W_1(y) \\ T_2: & R_2(y) & & W_2(y) & \end{array}$$

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 H_c^+-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:

$$R_1(y)\ W_1(y)\ R_2(y)\ W_2(y) \Rightarrow R_2(y)\ W_2(y)\ R_1(y)\ W_1(y)$$
$$R_1(y)\ W_1(y)\ R_2(y)\ W_2(y) \Rightarrow R_2(y)\ W_2(y)\ R_1(y)$$

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_c-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*.

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

$$R_1(y)\ W_1(y)\ R_2(y) \Rightarrow R_2(y)\ R_1(y)\ W_1(y)$$
$$R_1(y)\ W_1(y)\ R_2(y) \Rightarrow R_2(y)\ W_2(y)\ R_1(y)\ W_1(y)$$
$$R_1(y)\ W_1(y)\ R_2(y) \Rightarrow R_2(y)\ W_2(y)\ R_1(y)$$

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:

$$R_1(y)\ R_2(y)\ W_2(y) \Rightarrow R_2(y)\ W_2(y)\ R_1(y)$$

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 c_a-patterns should. Further, it is most natural to disallow an a_p-pattern - i.e. to have it forced into either an a_s- or a c_a-pattern. Also, it is quite natural to take notice of an a_s-pattern - i.e. to let the acquiring-sequence of article-instances matter.[8] Note that the c_a- and a_s-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,**

$$\begin{array}{llll} H_{DCA} = & & & \\ & T_1: R_1(x)W_1(x) & & R_1(y)\ldots \\ & T_2: & R_2(x)R_2(y)W_2(y) & \ldots, \end{array}$$

there are two c_a-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,**

$$\begin{array}{llll} H_{IRI} = & & & \\ & T_1: R_1(x)W_1(x) & & R_1(y)W_1(y)\ldots \\ & T_2: & R_2(x)R_2(y) & \ldots, \end{array}$$

there is one c_a-pattern and one c_b-pattern embedded contributing in reverse ways. The x-accesses (corresponding to the c_a-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,

$$T_1T_2,$$

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**,

$$
\begin{array}{lll}
H_{IR2} = & & \\
T_1: R_1(x) & & R_1(y) \ldots \\
T_2: & R_2(x)W_2(x)R_2(y)W_2(y) & \ldots ,
\end{array}
$$

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 $T_1 \rightarrow T_2$, and the y-accesses (corresponding to the c_a-pattern) contribute $T_2 \rightarrow T_1$. 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,

$$T_2T_1,$$

and we can allow such a schedule.
In the following schedule reflecting a **double check-before situation**,

$$
\begin{array}{lll}
H_{DCB} = & & \\
T_1: R_1(x) & & R_1(y)W_1(y) \ldots \\
T_2: & R_2(x)R_2(y)W_2(x) & \ldots ,
\end{array}
$$

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 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,

$$T_2T_1 \; / \; T_1T_2,$$

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_{IR1}/H_{IR2} 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 $WR\text{-}RW\text{-}WW_Y(H)$ and $WR\text{-}RW\text{-}WW_y(H)$ 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. $WR\text{-}RW\text{-}WW_Y(H)^*$ and $WR\text{-}RW\text{-}WW_y(H)^*$ represent the transitive closures of $WR\text{-}RW\text{-}WW_Y(H)$ and $WR\text{-}RW\text{-}WW_y(H)$ 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 $C_{L:WR\text{-}RW\text{-}WW}$ all the binary relations $WR\text{-}RW\text{-}WW_Y(H)^*$ or $WR\text{-}RW\text{-}WW_y(H)^*$ (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 $C_{L:WR\text{-}RW\text{-}WW}$, 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

$$R_1(y)...R_2(y) \Rightarrow R_2(y)...R_1(y)$$

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

$$R_1(y)...W_2(y) \Rightarrow W_2(y)...R_1(y)$$

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:

$$
\begin{array}{llllll}
H_{NFSS} = & & & & & \\
T_1: R_1(x_a)W_1(x_a) & & R_1(y_b)W_1(y_b) & & ... & \\
T_2: & R_2(x_a)R_2(y_b) & & R_2(y_c)W_2(y_c) ..., & &
\end{array}
$$

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_{NFSS}-schedule (which is a variant of the H_{IRI}-schedule from the "Basic Concurrency Control" section) will be allowed by our concurrency rules, and the topological ordering of WR_D* gives:

$$T_1T_2$$

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_{NFSS}-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 $RW_D(\text{Any } H)^*$ 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 $WR_D(\text{Any } H)^*$ 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 $WR_D(\text{Any } H)^*$ 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 $\text{WR-RW-WW}_{y/Y}(H)^*$. 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:

$$H_1 \notin C_{WR}$$

F

urther, 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 $R_1(y)$ occurs. After later *Rescheduling T_2*, the final schedule is:

H_2 =
T_1: $R_1(x)W_1(x)R_1(y)R_1(z)W_1(y)$
T_2: $R_2(z)R_2(y)R_2(x)W_2(z)$

An underlying assumption is that when $R_1(y)$ is requested and thus invokes an abortion of T_2, $R_1(y)$ 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.

Concurrency Control Mechanisms

We have for our combined local and global criteria:

$$H \in C_{WR} \text{ iff}$$

$$\bullet\ \forall\,(Y \textit{ is a Site})\left[WR - RW - WW_{Y}\left(\text{Any}_{H}\right)^{*} \textit{ is a Partial Order}\right]$$

$$\bullet\ WR_{D}\left(\text{Any}_{H}\right)^{*} \textit{ is a Partial Order}$$

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_D, WR-RW-WW_Y and WR-RW-WW_D[11] 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_{bc} 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_{bc}-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:

$$CSG\text{: None} \qquad PSG\text{: None}$$

$$CSG_{x} : T_1T_2 \qquad CSG_{y} : T_2T_1 \qquad CSG_{z} : T_2T_1$$

Thus we must conclude:

$$\left(H_1 \notin CSR\right) \text{ and } \left(H_1 \notin C_{WR}\right)$$

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

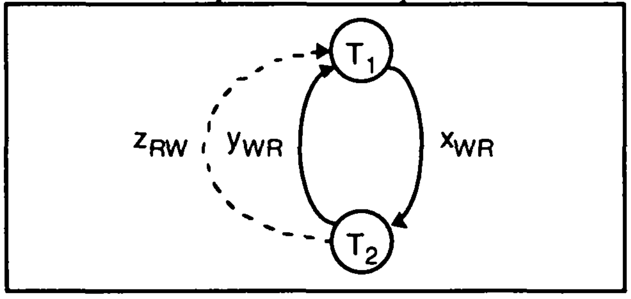

Figure 9. PSG(H_1) vs. CSG(H_1)

From Fig. 7 we have the following topological sorts:

$$CSG\text{: } T_1T_2 \qquad PSG\text{: } T_1T_2$$

$$CSG_{x} : T_1T_2 \qquad CSG_{y} : T_1T_2 \qquad CSG_{z} : T_1T_2$$

11. The binary relation WR-RW-WW_D(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:

$$\left(H_2 \in CSR\right) \text{ and } \left(H_2 \in C_{WR}\right)$$

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

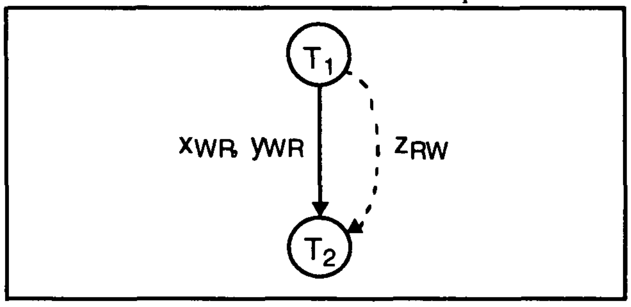

Figure 10. PSG(H_2) vs. CSG(H_2)

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. Alternatively 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/LockingUntilEnd 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_1-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.

Concurrency Control Examples

The H_1-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:

x:	y:
Quality = 95%	Quality = 90%
Price = 10 \$	Price = 5 \$
No-Left = 3	No-Left = 3
Who-Pur. = -	Who-Pur. = -

We look at the following three wander-transactions:

T_1: Acquire One: Lowest-Price & Quality $\geq$ 85 %
T_2: Acquire One: Price $\leq$ 15 \$ & Highest-Quality
T_3: Acquire One: Price $\leq$ 15 \$ & Highest-Quality
+ Acquire One: Lowest-Price & Quality $\geq$ 85 %

A possible schedule corresponding to wander-transactions T_1 and T_2 is:

$$H_3 = \begin{array}{lllllll} T_1: & R_1(x) & & R_1(y) & & W_1(y) & \\ T_2: & & R_2(x) & & R_2(y) & & W_2(x) \end{array}$$

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

$$\begin{array}{ll} CSG\text{: None} & PSG\text{: } T_1T_2 \ / \ T_2T_1 \\ CSG_x: T_1T_2 & CSG_y: T_2T_1 \end{array}$$

Thus we may conclude:

$$\left(H_3 \notin CSR\right) \text{ but } \left(H_3 \in C_{WR}\right)$$

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 H_3 may also be allowed by the lock-setting mechanism.

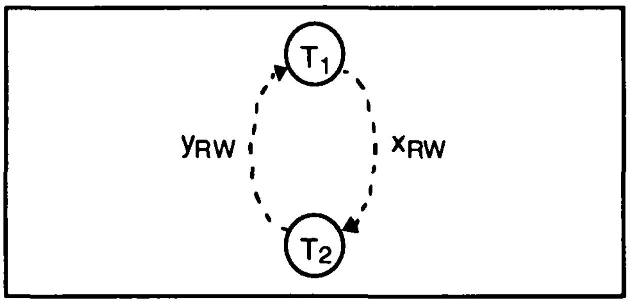

Figure 11. PSG(H_3) vs. CSG(H_3)

A possible schedule corresponding to wander-transactions T_1 and T_3 is:

$$H_4 = \begin{array}{lll} T_1: & R_1(x) & & R_1(y)W_1(y) \\ T_3: & & R_3(x)R_3(y)W_3(x)W_3(y) & \end{array}$$

This schedule reflects **one inconsistent retrieval situation**, see the "Basic Concurrency Control" section. From Fig. 9 we have the following topological sorts:

$$\begin{array}{ll} CSG\text{: None} & PSG\text{: } T_3T_1 \\ CSG_x : T_1T_3 & CSG_y : T_3T_1 \end{array}$$

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:

T_1: Acquire One: Price ≤ 7.5 \$ & Highest-Quality

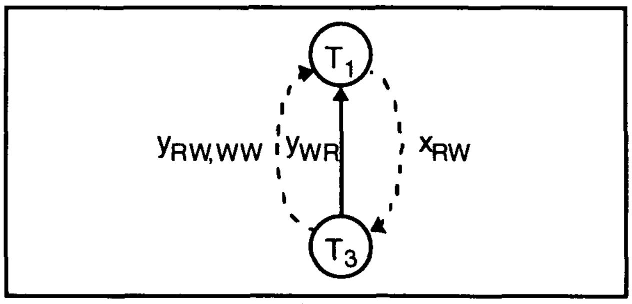

Figure 12. PSG(H_4) vs. CSG(H_4)

A possible schedule corresponding to wander-transactions T_1, T_2 and T_3 is:

$$\begin{array}{lllllll} H_5 = & & & & & & \\ T_1: & R_1(x) & & & & R_1(y)W_1(y) & \\ T_2: & & & R_2(x)R_2(y) & & & W_2(x) \\ T_3: & & R_3(x)R_3(y)W_3(x) & & W_3(y) & & \end{array}$$

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:

$$\begin{array}{ll} CSG\text{: None} & PSG\text{: } T_3T_1T_2 \,/\, T_3T_2T_1 \\ CSG_x : T_1T_3T_2 & CSG_y : T_2T_3T_1 \end{array}$$

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_3 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.

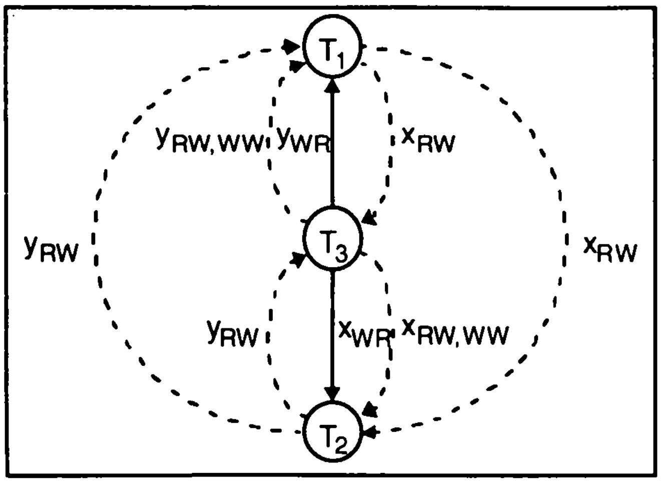

Figure 13. PSG(H_5) vs. CSG(H_5)

Once more, if x and y had belonged to a unique site instead of to two different sites, the situations reflected in H_3, H_4 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_r^+ 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_D(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

Bancilhon, F., Kim, W. and Korth, H.F. (1985) A Model of CAD Transactions, **11th Int. Conf. on Very Large Data Bases**.

Beeri, C., Bernstein, P.A. and Goodman, N. (1986) A Model for Concurrency in Nested Transaction Systems, **Technical Report TR-86-03**, Wang Institute.

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.

Bernstein, P.A., Hadzilacos, V. and Goodman, N. (1987) **Concurrency Control and Recovery in Database Systems**, Addison-Wesley.

Bhargouti, N.S. and Kaiser, G.E. (1991) Concurrency Control in Advanced Database Applications, **ACM Computing Surveys, 23(3)**.

Breitbart, Y., Garcia-Molina, H. and Silberschatz, A. (1992) Overview of Multidatabase Transaction Management, **VLDB Journal, 1(2)**.

Du, W. and Elmagarmid, A.K. (1989) Quasi Serializability: A Correctness Criterion for Global Concurrency Control in InterBase, **15th Int. Conf. on Very Large Data Bases**.

Elmagarmid, A.K. (ed.) (1992) **Database Transaction Models for Advanced Applications**, Morgan Kaufmann.

Garcia-Molina, H. and Wiederhold, G. (1982) Read-Only Transactions in a Distributed Database, **ACM Trans. on Database Systems, 7(2)**.

Garcia-Molina, H. and Kogan, B. (1988) Achieving High Availability in Distributed Databases, **IEEE Trans. on Software Engineering, 14(7)**.

Gray, J.N., Lorie, R.A., Putzolu, G.R. and Traiger, I.L. (1976) "Granularity of Locks and Degrees of Consistency in a Shared Data Base" In Nijssen, G.M. (ed.) **Modelling in Data Base Management Systems**, IFIP TC-2 Working Conference, North-Holland.

Ibaraki, T., Kameda, T. and Minoura, T. (1987) Serializability with Constraints, **ACM Trans. on Database Systems, 12(3)**.

Korth, H.F. and Speegle, G.D. (1988) Formal Model of Correctness Without Serializability, **ACM SIGMOD Int. Conf. on Management of Data**.

Nygård, M. (1993a) Relaxed Criteria for Concurrency Control in Distributed Databases, **IFIP TC6'93 - Int. Symp. on Network Information Processing Systems**, Sofia - Bulgaria, 23 - 54.

Nygård, M. (1993b) Sensible Variants of Non-Serializability in Replicated Distributed Databases, **IFIP TC6'93 Int. Symp. on Network Information Processing Systems**, Sofia - Bulgaria, 55 - 73.

Nygård, M. (1994a) Tailor-Made Concurrency Control with Specialized Distributed Transactions, **ACIS'94 - 5th Australasian Conf. on Information Systems**, Melbourne - Australia, 355 - 376.

Nygård, M. (1994b) Partial Recoverability with Distributed Transactions, **IEEE IPCCC'94 - 1994 Int. Phoenix Conf. on Computers and Communications**, Phoenix - USA, 68 - 76.

Papadimitriou, C. (1986) **The Theory of Database Concurrency Control**, Computer Science.

Schlageter, G. (1978) Process Synchronization in Database Systems, **ACM Trans. on Database Systems, 3(3)**.