References

[BGRS91] Y. Breitbart, D. Georgakopolous, M. Rusinkiewicz, and A. Silberschatz. On rigorous transaction scheduling. *IEEE Transactions on Software Engineering*, 1991.

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

[BST90] Y. Breitbart, A. Silberschatz, and G. R. Thompson. Reliable transaction management in a multidatabase system. In *Proceedings of ACM-SIGMOD 1990 International Conference on Management of Data, Atlantic City, New Jersey*, pages 215–224, 1990.

[Lam78] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[LL90] G. Le Lann. Critical issues for the develeopment of distributed real-time computing systems. Technical Report 1274, Institut National de Recherche en Informatique et en Automatique, April 1990.

[MRB+92] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. The concurrency control problem in multidatabases: Characteristics and solutions. In *Proceedings of ACM-SIGMOD 1992 International Conference on Management of Data, San Diego, California*, 1992.

[Pap79] C. Papadimitriou. Serializability of concurrent database updates. *Journal of the ACM*, 26(4):631–653, October 1979.

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

[SKS91] N.R. Soparkar, H.F. Korth, and A. Silberschatz. Failure-resilient transaction management in multidatabases. *IEEE Computer*, December 1991.

[Son88] S.H. Son, editor. *ACM SIGMOD Record: Special Issue on Real-Time Databases.* ACM Press, March 1988.

operations that coincide in the mapping imposed by $time_i$ can be made to be distinct. These, and other extensions, are not further investigated in any detail in this paper.

7.3 Additional Practical Considerations

Let us direct attention to some additional practical considerations. First, it is necessary for the LTM_i to explicitly, or implicitly, provide to the $MDBS_i$ the allocated points in $time_i$ for the *synch* operations. One way in which this is achieved is to restrict the LTM_i schedules to be those where the *synch* operations may be associated with some pre-determined operation of each transaction in the sense that the *synch* operation should be executed directly after that LTM_i operation, and before the next LTM_i operation to which is mapped a *synch* operation. Such an LTM_i operation is referred to as a serialization function in [MRB+92]. Second, it is important that the $time_i$ mapping for a *synch* operation should remain unchanged once it is made known to $MDBS_i$ by the LTM_i — which we may refer to as the *stability* of the *synch* operations. The reason why this stability is needed is because the $MDBS_i$ may execute the synchronization protocol corresponding to the mapping by $time_i$ of a particular *synch* operation, and if this mapping changes subsequently, the global serializability may be jeopardized.

8 Conclusions

Ensuring the serializability of transaction executions in an environment consisting of several autonomous sites was investigated in detail in this paper. We provided a basis for achieving the synchronization among the subtransactions executing at several sites in order to achieve global serializability. In doing so, we provided the conditions necessary on the schedules at local sites to permit such synchronization. The classes of concurrent schedules were delineated as to their viability with regard to achieving globally serializable schedules. Our research indicates which existing or future protocols are amenable to synchronization, and also suggests the effective means to do so. Thus, we provide a characterization for integrating the transaction schedules arising from autonomous sites.

7 Further Observations

In this section, we consider some common concurrency control protocols for the local transaction systems in the context of the above discussions. Also, we consider variations of the transaction model that we have described above, and examine the implications in the context of MDBSs.

7.1 Common Protocols

We continue to use the restricted model of transactions described in Section 6 for the following discussion as well. From Theorem 4, it should be clear that schedules from any subclass of Q may also be synchronized with the synchronization protocol executing under the pragmatic restriction. Hence, the class of 2PL schedules, and the class of *timestamp* (TS) schedules are amenable to such synchronization.

Theorem 2 indicates that not all schedules from the *strictly serializable* (SSR) class (e.g., see [Pap79]) can be synchronized. In fact, for the synchronization of SSR schedules (e.g., see [BGRS91]), note that the *synch* operations are guaranteed to occur within the requisite active intervals by ensuring that the corresponding subtransactions do not execute concurrently. In any case, since the entire SSR class cannot have an efficient scheduler in the (e.g., see [Pap79, Pap86]), this discussion may not be very relevant. Note that not all the schedules of the class P3 that are produced by protocol used in SDD-1 systems (e.g., see [Pap79]), are amenable to synchronization either.

Example 6. The schedule $R_3[b]R_1[b]W_1[a]R_2[a]W_3[a]W_2[a]$, which is a modified version of Example 3, is SSR in the restricted model with the equivalent serial schedule being $< T_3T_1T_2 >$. This is an example of an SSR schedule, which is not CSR, and hence, does not permit synchronization in the order of execution of the conflicting operations. Note that the the schedule of Example 4 is also SSR. □

7.2 Variants of the Model

The imposition of restrictions on the transactions helps to further classify the protocols that may be synchronized. For example, as discussed in [Pap79], the absence of *blind* writes in the transactions changes the containment structure among the classes. In such cases, it is possible to synchronize the entire class VSR by Theorem 3, and the entire class SSR even under the pragmatic restriction.

The discussions that have been limited to the restricted model of transactions described in Section 5 may be extended to a more general model. For example, the major observation of the importance of the class Q of schedules, as described by Theorem 4, could be easily extended by requiring the real numbers r_j to lie in the active intervals of the transactions in the general model. Similarly, it should be clear that extending these ideas to *partial* orders of transactions, and partially ordered local schedules, is not a difficult task. The proof of Theorem 3 suggests the way in which

the subsequent possibility of aborting T_j in case the synchronization fails, is precluded. Note that since the LTM_i are unaware of the distinction between a local transaction and subtransaction, this requirement must extend to all the transactions executing at a site.

Example 4. The schedule $R_3[a]R_1[b]W_1[a]R_2[c]W_2[d]W_3[c]$, which is CSR, has the sole equivalent serial schedule $< T_2T_3T_1 >$. However, note that $synch_1$ and $synch_2$ cannot lie within their respective active intervals for the same $time_i$ mapping. This happens because the active intervals for T_1 and T_2 do not overlap, and also, the order of occurrence of these intervals is different as compared to the serialization order of the corresponding transactions. □

We now consider the restricted model of transactions that have exactly two operations, $R[\ldots]$ followed by $W[\ldots]$, each (e.g., see [Pap79]). In this model, the two operations that constitute each transaction correspond to the first and the last operations, clearly. For the pragmatic restriction in the CSR schedules, the proof of Theorem 3 suggests that we should define a class of schedules with the real number r_j to additionally lie within the active interval in $time_j$ for the transaction T_j. By setting the $time_i$ mapping as is done in the proof of Theorem 3, we ensure that $synch_j$ will lie within the active interval for T_j. The construction of the function $time_i$ for the proof of Theorem 3 indicates that without loss of generality, we may require the numbers r_j to be distinct non-integral, reals. Note that the *endsynch* operations should also get executed during the active interval to permit the abort of a subtransaction in case the synchronization fails.

Example 5. The schedule $R_1[a]R_2[a]R_3[b]W_1[b]W_2[cd]W_3[c]$, which has the equivalent serial schedule $< T_2T_3T_1 >$, provides an example of a CSR schedule that permits the pragmatic restriction. Note that all serial schedules obviously permit this restriction. □

Thus, we have characterized a class of schedules for which the set of distinct real numbers $r_1, r_2, \ldots, r_m$ exist with the properties as described for CSR, and additionally, they have the pragmatic restriction that for each transaction T_j, it is the case that $time_i(R_j[\ldots]) < r_j < time_i(W_j[\ldots])$. These schedules form the class Q in [Pap79], and they may be characterized by a graph very similar to an SG. The prefix class for Q is efficiently recognizable, and hence, there exist efficient schedulers for the class.

Theorem 4. *The class of CSR schedules of LTM_i that can be synchronized to provide globally serializable schedules using the synchronization protocol executed under the pragmatic restriction includes, and is no larger than, Q.*

Proof: By Theorem 1, the definition of the class Q, and methods similar to the proof for Theorem 3. □

Theorem 3. *The class of LTM_i schedules that can be synchronized to provide globally serializable schedules includes the class CSR.*

Proof: Firstly, Theorem 1 provides the basis for using the synchronization protocol. Next, we define a $time_i$ function corresponding to any CSR schedule that is plausible for the synchronization protocol to have acted upon. Note that in the following construction, the CSR properties of the LTM_i schedule remain unaffected.

Let $time_i$ map each $synch_j$ operation to the number r_j. Change the $time_i$ function by separating any set of LTM_i and *synch* operations mapped to the same value by small ϵ differences without affecting the precedence relationship of the allocated points. This separation is ensures that $time_i$ is a bijection, and hence, that the $MDBS_i$ does not need to execute more than one *synch* operation simultaneously. Hence, directly after each *synch* operation, and before the next point allocated in $time_i$, specify the corresponding *endsynch* operation. The last addition ensures that no two synchronization intervals overlap at site S_i.

Since the possibility of aborts of transactions leaves the validity of the r_j numbers unaffected, we note that the synchronization is indeed possible. □

6 A Pragmatic Restriction

In this section, we restrict our attention to a smaller subclass of the CSR schedules to ensure that the synchronization protocol can be utilized effectively in practice. This subclass of schedules is especially useful in an online transaction system. The subclass is one where we restrict the position of the $MDBS_i$ operations in $time_i$ with respect to the LTM_i operations. Notice that in Section 5, the position of the *synch* operation with respect to the time that the corresponding subtransaction was actually in execution, was not restricted. We shall make the necessary restrictions in this regard more precise in this section.

We define the *active interval* of a transaction to consist of the interval (i.e., in $time_i$, or in the local schedule at site S_i) between its first operation and its last operation. The synchronization protocol is said to execute under a *pragmatic* restriction if each $synch_j$ operation lies within the active interval of the corresponding subtransaction T_j. The justification for this restriction is that if $synch_j$ precedes the first operation of the subtransaction, then it happens to be the case that it is possible to synchronize the global transactions without the LTM_i even being aware of the existence of the subtransaction. This is possible only in situations where the subtransactions that will execute at a site are known *a priori*. Furthermore, suppose that the $synch_j$ operation occurs after the last operation of T_j. For models where T_j is considered committed when the last operation is executed (e.g., an extended model of [Pap79] regards the last operation to be a W operation that writes into a data item that is specific to the transaction in question — so as to indicate a commit),

the sequence of operations in a restricted schedule represents the $R[\ldots]$ operations as accessing the values of data items that were last changed by a $W[\ldots]$ operation from a transaction which was not aborted. If there was no preceding $W[\ldots]$ of an unaborted transaction, then the values are those that were present before the schedule was executed. Also, note that the $R[\ldots]$ operations of the aborted transactions are not of significance, and therefore, can be safely omitted by the restriction.

Example 3. Consider the schedule $W_1[a]R_2[a]W_3[a]W_2[a]$ which is VSR with the equivalent serial schedule in the order $< T_3T_1T_2 >$. If transaction T_1 is aborted, which could happen subsequently due to an atomic commitment protocol, then the above schedule is no longer equivalent to either $< T_2T_3 >$, or $< T_3T_2 >$. □

Lemma 1. *Consider an MDBS where each local schedule generated is VSR. If arbitrary transaction aborts are supported, then each LTM_i supports a class that is no larger than CSR.*

Proof: Consider a schedule h with operations from a set τ of transactions. Let $h_{\tau'}$ denote the same schedule, but with all transactions aborted that do not belong to τ', where $\tau' \subset \tau$. The corresponding schedule restricted to the transactions in τ' is denoted by $h_{\tau'}$. Since an arbitrary subset of transactions in a schedule may be aborted, there is a need to consider arbitrary $\tau' \subset \tau$ that create the restricted schedules $h_{\tau'}$. A known result (e.g., see the discussion on the *monotonicity* of CSR in [Pap86]) indicates that h is CSR if, and only if, each $h_{\tau'}$ is VSR. □

Note that two equivalent CSR schedules remain equivalent when both are restricted to arbitrary transactions. This follows from the fact that the SG for a restricted schedule is simply a subgraph of the SG for the same schedule without the restriction.

Theorem 2. *The class of VSR schedules of LTM_i that can be synchronized by the synchronization protocol is no larger than CSR.*

Proof: By the requirements of the synchronization protocol, the atomicity of the transaction executions, and Lemma 1. □

5 Synchronization of CSR Schedules

Consider a schedule consisting of m transactions, $T_1, T_2, \ldots, T_m$. A characterization of a CSR schedule with m transactions is that there exist m real numbers $r_1, r_2, \ldots, r_m$ such that r_i is associated with transaction T_i, and the numbers have the following property. Consider two transactions T_i and T_j that have conflicting operations. For each pair of conflicting operations between these two transactions, if the operation from T_i precedes the one from T_j in the schedule, then $r_i < r_j$. Note that one such set of real numbers can be obtained by the topological sort of the SG. Also from the SG, it should be evident that aborts of arbitrary transactions do not affect the validity of these numbers. We use this characterization by numbers to prove the following result.

consider the operations, $send_{x_1 i_1}$, $endsynch_{x_1 i_1}$, $synch_{x_2 i_1}$, $recv_{x_2 i_1}$, $send_{x_2 i_2}$, $endsynch_{x_2 i_2}$, ..., $synch_{x_1 i_u}$, $recv_{x_1 i_u}$, $send_{x_1 i_1}$, in that order (the property relating the $time_{i_k}(synch_{x_j i_k})$ values as described above provides the clue to this order of operations). Figure 3 gives a simple example of this order (marked by numbers 1, 2, ..., 8, 1) for two global transactions, T_x and T_y, executing at sites S_p and S_q. By the description of the synchronization protocol, each adjacent pair of operations in this order is related by the *happened-before* relationship (e.g., see [Lam78]). However, that is not possible since the order described constitutes a cycle, whereas the *happened-before* relationship defines a partial order on the operations. Hence, such a cycle in the global SG is not possible. □

In the subsequent sections, we restrict attention to schedules at a particular site S_i. It should be clear that any local schedule with *synch* operations which have the properties as described above, are plausible local schedules for which there exist global executions that are serializable. However, other considerations may restrict the allowable sets of schedules as we shall examine below.

4 Effects of Transaction Failures

The synchronization mechanism described above requires the use of an abort operation to ensure global serializability. In fact, any synchronization protocol that relies on checking for the correctness of the executions must have the option of aborting subtransactions whenever the global serializability is threatened, since the LTM_i are not under the control of the $MDBS_i$. Hence, the underlying LTM_i must support the aborts of arbitrary transactions that execute under its control. Note that we have not considered the question of ensuring the *atomicity* of the global transactions (e.g., see [SKS91]) in the sense that all or none of the subtransactions of a global transaction should be completed. This is the problem of global atomic commitment which is effected by means of committing, or aborting, all the subtransactions. In this paper, we have considered the synchronization protocol to be distinct as compared to a global atomic commitment protocol (in standard distributed database systems, the two are inseparable — e.g., see [SKS91]). However, it should be noted that the latter would have to be executed, and as a consequence, even after a subtransaction is fully executed at a site in terms of all the operations as described in Section 2, it may need to be aborted.

While we do not formally include an abort operation in the repertoire of operations that the LTM_i supports, we nevertheless describe the effect of the *abort* of transactions on a local schedule. Since it is the case that a schedule with aborted transactions should be such that the operations of those transactions should not have executed at all, consider a schedule to be restricted to only the operations of transactions that are not aborted. Note that the *committed projection* used to check that a schedule is CSR or VSR is exactly such a restricted schedule (e.g., see [BHG87]). That is,

Theorem 1. *The synchronization protocol ensures globally serializable executions.*

Proof: We exhibit that the serializability graph constructed for the global transactions will indeed be acyclic. Assume that is not so, and that a cycle of global transactions $T_{x_1} \rightarrow T_{x_2} \rightarrow \ldots \rightarrow T_{x_u}$ exists, and without loss of generality, assume that the transactions in the cycle are distinct. Then the following situation must exist. There are sites S_{i_1}, S_{i_2}, ..., S_{i_u}, not all the same,[1] where the subtransaction pairs $(T_{x_1 i_u}, T_{x_1 i_1})$, $(T_{x_2 i_1}, T_{x_2 i_2})$, ..., $(T_{x_u i_{u-1}}, T_{x_u i_u})$ corresponding to global transactions T_{x_1}, T_{x_2}, ..., T_{x_u}, respectively, execute. Note that a subtransaction $T_{x_j i_k}$ executes at the site S_{i_k}, and that it is distinct from the other transactions since we assume that any given global transaction has at most one subtransaction executing at a particular site. These subtransactions must cause the cycle described above. Using the definition of a global SG, we have $time_{i_1}(synch_{x_1 i_1}) < time_{i_1}(synch_{x_2 i_1})$, $time_{i_2}(synch_{x_2 i_2}) < time_{i_2}(synch_{x_3 i_2})$, ..., $time_{i_u}(synch_{x_u i_u}) < time_{i_u}(synch_{x_1 i_u})$.

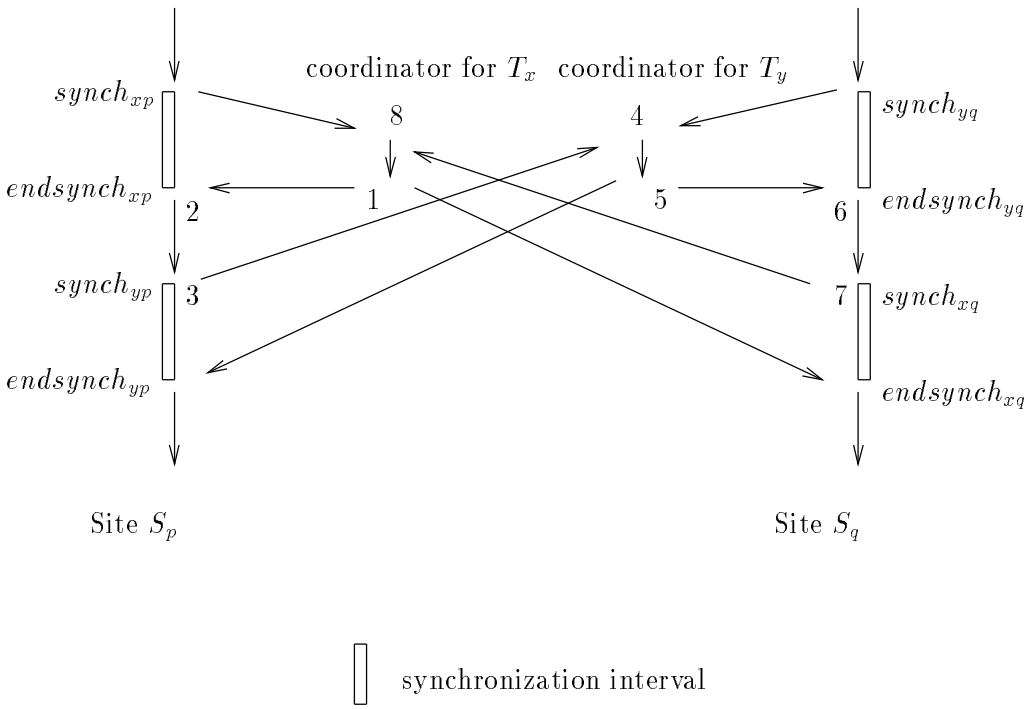

Figure 3: Impossible Cycle with the Synchronization Protocol

At the coordinator site S_{x_j} for a global transaction T_{x_j}, let the receipt operation of the message sent by site S_{i_k} due to the $synch_{x_j i_k}$ operation corresponding to the subtransaction $T_{x_j i_k}$, be denoted by $recv_{x_j i_k}$. Similarly, let the send operation of the message to site S_{i_k} which will be received by the $endsynch_{x_j i_k}$ operation corresponding to the subtransaction $T_{x_j i_k}$, be denoted by $sendx_j i_k$. Now

[1] Local concurrency control ensures that global cycles are not caused due to the executions at a single site alone.

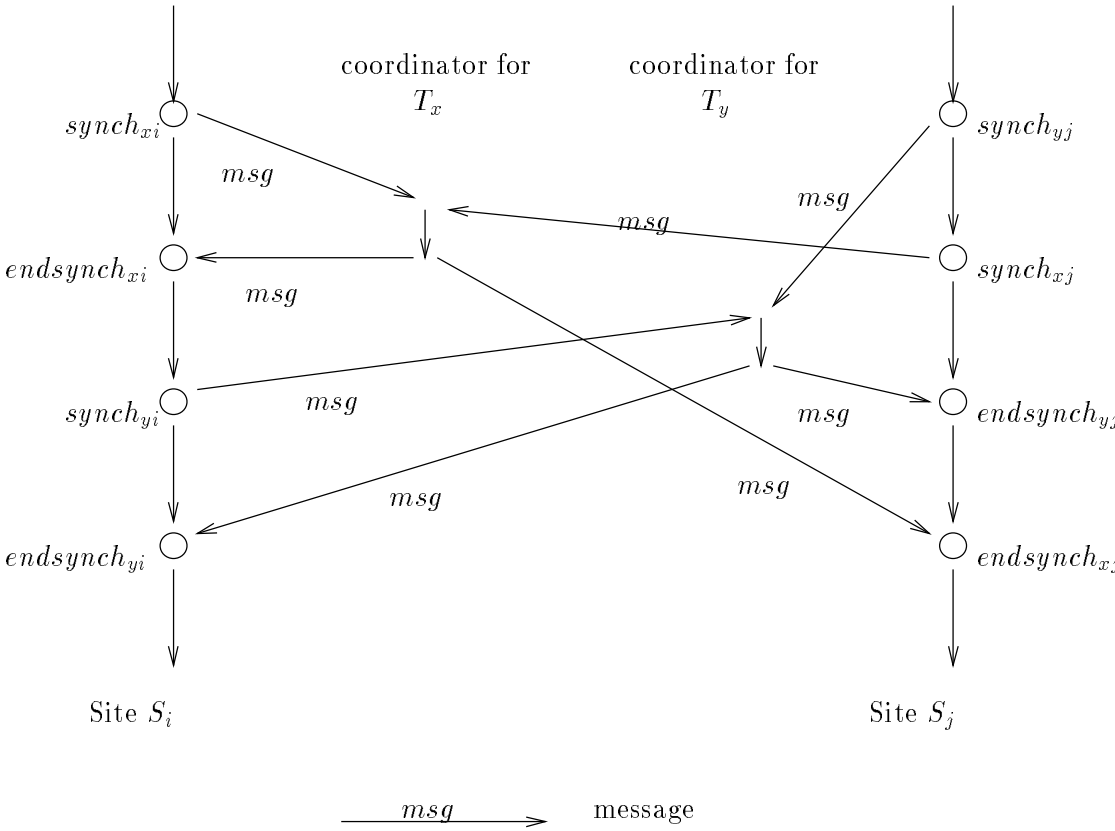

Figure 2: Overlapped synchronization intervals

involved in a non-serializable execution. Hence, such a global transaction must be aborted by effecting the aborts of each of its subtransactions. We discuss the implications of aborts on the allowable local schedules in Section 4. This completes the description of the entire synchronization protocol.

Example 2. Consider an MDBS where each local DBMS uses the 2PL protocol, and the two-phase commit protocol (e.g., see [BHG87]) is used to coordinate the execution of the global transactions. To ensure global serializability, each subtransaction is required to hold all its locks until the point that it is committed (e.g., see [SKS91]). The *synch* operation may occur at any point after all the locks for a particular subtransaction are procured, but before the subtransaction is committed. The *endsynch* operation, which occurs after the *synch* operation, must occur prior to the commitment. It can be shown that the position of the *synch* operation in the local schedule does have the properties as described above. □

nodes, and an edge from a node T_j to T_k that exists if they have corresponding subtransactions T_{ji} and T_{ki}, respectively, that execute at a site S_i, and $time_i(synch_{ji}) < time_i(synch_{ki})$. The acyclicity of such a graph implies the requisite total order that may be obtained by a topological sort of the nodes.

The synchronization protocol, which is executed once for every global transaction T_x, can now be described. In its basic form, the protocol is similar to the ones described in [MRB+92, SKS91]. The protocol requires that a coordinator site for the transaction T_x should receive the messages sent for synchronization from the participating sites. Upon receiving all such messages, the coordinator sends an acknowledging message to each participating site to complete the protocol. Our protocol differs from the other protocols in a number of ways. First, it permits a different coordinator site to be used for each global transaction. Second, and more importantly, the point in $time_i$ that a site S_i sends a synchronizing message to the coordinator is not necessarily related to any specific operation in the LTM_i (e.g., the *serialization function* used in [MRB+92] defines a specific LTM_i operation — directly after which, the synchronization message must be sent). This generality provides us the means to study the classes of local schedules that may be synchronized in this manner, as will be discussed below.

Thus, our synchronization protocol for a global transaction T_x executes in the following manner. At each site S_i at which a subtransaction T_{xi} of the global transaction T_x executes, $MDBS_i$ sends a message to S_x, the coordinator site for T_x, at the $synch_{xi}$ operation for T_{xi}. After $MDBS_x$ at the coordinator site S_x collects these messages from all the subtransactions corresponding to T_x, it sends an acknowledging message to each participating site. The receipt of such a message for subtransaction T_{xi} by $MDBS_i$ is regarded as the $endsynch_{xi}$ operation.

Define the *synchronization interval* for a subtransaction T_{xi} to be the interval (i.e., in $time_i$, or in the local schedule at site S_i) between $synch_{xi}$ and $endsynch_{xi}$ operations. If two subtransactions T_{xi} and T_{yi} that execute at a site S_i need to be serialized in a particular order (e.g., as may happen if they both update a common data item), then their corresponding synchronization intervals should not overlap. This is a necessary condition for the correctness of the synchronization protocol. In Example 1 below, we show a non-serializable execution that could occur if the intervals do overlap.

Example 1. A non-serializable execution of two global transactions T_x and T_y is depicted in Figure 2 when the synchronization intervals of their corresponding subtransactions overlap at one of their common execution sites. In the figure, T_x is serialized before T_y at site S_i, and T_y is serialized before T_j at site S_j. The messages sent between the sites, as depicted in the figure, correspond to the messages of the synchronization protocol. □

If the protocol fails to execute as described above (e.g., if the coordinator does not receive the messages from all the participating subtransaction sites), then the global transaction may be

$time_i(synch_k)$ where $time_i(synch_j) = r_j$, and $time_i(synch_k) = r_k$, and for each $synch_l$, $1 \leq l \leq m$, $time_i(R_l) < time_i(synch_l) < time_i(W_l)$ holds.

- 2PL: In addition to satisfying the constraints of the Q class, if $time_i(R_j) < time_i(W_k)$, where R_j and W_k conflict, then $r_j < r_k$, and if $time_i(W_j) < time_i(W_k)$, where W_j and W_k conflict, then $time_i(W_j) < r_k$, must both hold.

- TS: In addition to satisfying the constraints of the Q class, if $time_i(R_j) < time_i(R_k)$, then $r_j < r_k$.

- P3: Consider SG′, the undirected SG (e.g., see [BHG87]) for a schedule. Define a cycle to be a sequence $(T_{j_1}, T_{j_2}, \ldots, T_{j_m})$ for $m > 1$, such that adjacent elements in the sequence are adjacent in SG′, and also, T_{j_1} is adjacent to T_{j_m} in SG′. A cycle is *bad* if there is a common data item written by T_{j_1} and accessed by T_{j_m}, and a common data item written by T_{j_2} and read by T_{j_1}. The schedule is in P3 if for every bad cycle, it is not the case that $time_i(R_{j_1}) < time_i(W_{j_2}) < time_i(W_{j_1})$.

We do not provide the definitions of the possible extensions to transaction models that are more general than the restricted model used in [Pap79] — although the results in this paper also hold for the extensions unless specified otherwise.

3 Synchronization of the Local Schedules

In this section, we describe a synchronization protocol to ensure globally serializable executions. The protocol uses the *synch* operations mentioned above, and has several similarities with existing distributed database atomic commitment protocols (e.g., see [BHG87, SKS91]).

We require that for subtransactions T_{ji} and T_{ki} executing at site S_i, there must exist operations $synch_{ji}$ and $synch_{ki}$ such that if $time_i(synch_{ji}) < time_i(synch_{ki})$, then T_{ji} precedes T_{ki} in an equivalent serial schedule for the executions at that site. In other words, the order of execution of the *synch* operations in a local schedule provides a serialization order for their corresponding transactions. The schedule generated by LTM_i determines these points in $time_i$, and since the LTM_i does not differentiate between subtransactions and local transactions, it is the case that such *synch* operations need to be present for all transactions executed at a site.

As suggested in [MRB+92], to ensure globally serializable schedules, it is sufficient to ensure a total order on the global transactions as follows. For two subtransactions T_{ji} and T_{ki} executed at a site S_i, if $time_i(synch_{ji}) < time_i(synch_{ki})$, then for the corresponding global transactions, T_j precedes T_k in the total order on the global transactions. A graph similar to the *serializability graph* (SG) for CSR schedules (e.g., see [BHG87]) may be constructed with the global transactions as the

and this order subsumes the total order on the operations within each transaction. We assume that each LTM_i generates only VSR schedules, which is the most general correctness class given that an LTM_i only has syntactic information regarding the transactions (e.g., see [Pap86]).

It is useful to regard a schedule as a function, $time_i$, on the operations, where each operation from the set of m operations in a schedule is bijectively mapped to the natural numbers $\{1, 2, \ldots, m\}$. Since we are interested in the order of the operation executions, without loss of generality, this mapping may be regarded as the execution point in time as recorded at the site S_i.

To synchronize the execution of global transactions, the $MDBS_i$ must send and receive synchronization messages, which we denote by *synch* and *endsynch* operations, respectively. For each subtransaction T_k executing at the site S_i, we denote the unique pair of these operations by $synch_k$ and $endsynch_k$. Since the LTM_i is assumed to be unaware of these message exchanges, these operations are not part of the schedule generated by it. However, we can discuss the execution of these $MDBS_i$ operations by extending the $time_i$ mapping to include them. We regard each $MDBS_i$ operation to be mapped to a positive real number by the $time_i$ function. We assume that there is a single processor at each site S_i to ensure that $time_i$ remains a bijection (since each operation, irrespective of whether it is an $MDBS_i$ or an LTM_i operation, is then mapped to a distinct real number by $time_i$). Furthermore, all such real numbers that have a pre-image in the operations executed by LTM_i or $MDBS_i$, are referred to as points *allocated* in $time_i$. Note that the *happened-before* relation in [Lam78], for a particular site, is defined by the $time_i$ mapping.

We now provide the following definitions for some of the schedule classes to be discussed in terms of the $time_i$ mapping, and the LTM_i and $MDBS_i$ operations. Note that these are essentially obtained from existing definitions (e.g., see [Pap79], and we provide the necessary descriptions of the acronyms in the text. The set of transactions is taken to be $T_1, T_2, \ldots, T_m$, and for the ease of presentation, we adopt a restricted model where each transaction consists of exactly one read operation followed by a write operation (also, see Section 6). Below, we denote an arbitrary operation by the symbol O.

- CSR: There exist real numbers $r_1, r_2, \ldots, r_m$ such that if any two LTM_i operations O_j and O_k, $j \neq k$, conflict, and $time_i(O_j) < time_i(O_k)$, then $time_i(synch_j) < time_i(synch_k)$ where $time_i(synch_j) = r_j$, and $time_i(synch_k) = r_k$.

- SSR: There exists an equivalent serial schedule with the total order, $<_s$, such that if $time_i(W_j)$ $time_i(W_k)$ then in the serial schedule, $W_j <_s W_k$ holds.

- Q: There exist distinct, non-integral real numbers $r_1, r_2, \ldots, r_m$ such that if any two LTM_i operations O_j and O_k, $j \neq k$, conflict, and $time_i(O_j) < time_i(O_k)$, then $time_i(synch_j) <$

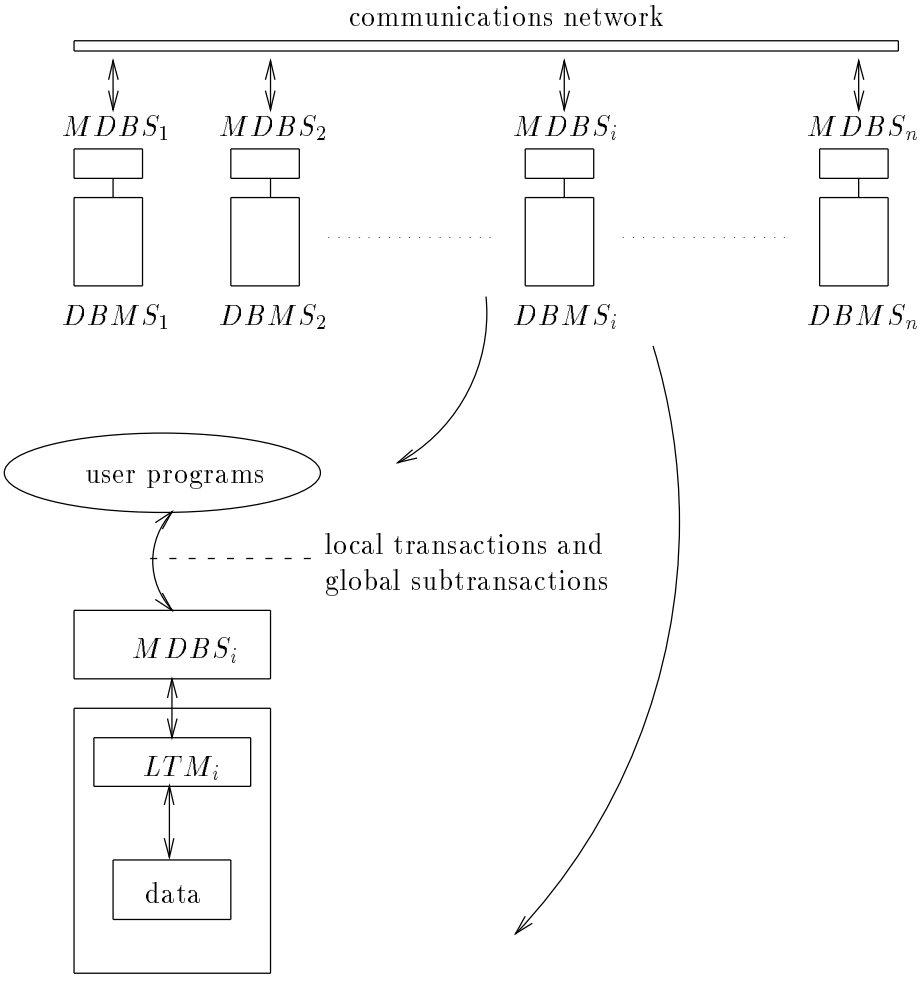

Figure 1: MDBS Structure

1. $R[\ldots]$ which reads the value(s) of the data items, mentioned within the brackets.

2. $W[\ldots]$ which writes the value(s) of data items, mentioned within the brackets.

Operations are assumed to execute atomically and instantaneously. Two operations from different transactions are said to *conflict* if they access a common data item and if one of them is a write operation.

A *transaction* consists of a totally ordered set of one or more such operations. Subscripts are used to indicate the particular transaction to which the operation belongs. For example, R_j denotes a read operation transaction T_j. To improve readability, whenever it is clear from the context, we omit subscripts.

Each LTM_i executes all the operations, and no other, from a set of transactions submitted to it in a sequence called a *schedule*. That is, the LTM_i imposes a total order, $<_i$, on the operations,

to guarantee globally serializable executions. Since any synchronization protocol may require that subtransactions be aborted, we study the effect of aborts on local schedules, and show that only the class of CSR schedules generated locally are permissible in an MDBS environment. Turning our attention to online transaction systems (i.e., systems where the transactions are not known *a priori*), we prove that a further restriction on the local CSR schedules is necessary in order to use the synchronization protocol. Using our characterization of the allowable classes of local schedules, which provides a measure of the permissible degrees of concurrency in the schedules, we examine several common concurrency control mechanisms that may be used at each local site in an MDBS environment.

The remainder of the paper is organized as follows. Section 2 provides the logical architecture and the notation for the system under consideration, and we use them to present our results. Section 3 describes a general synchronization protocol, with distributed coordinators, that is used in conjunction with each global transaction. The manner in which the synchronization protocol restricts the class of schedules generated at the local sites to be CSR, is discussed in Section 4. The means to synchronize the class of CSR schedules from each local site is described in Section 5. In Section 6, we study the subclass of CSR schedules at the local sites that may be synchronized while the concerned subtransactions are active. Section 7 discusses the feasibility of synchronizing some common classes of local schedules produced by existing concurrency control protocols. The section also provides discussions regarding the effect of some useful restrictions and extensions to our transaction model on the local schedules in the context of an MDBS environment. Finally, Section 8 constitutes the conclusions.

2 System Structure and Notation

An MDBS consists of n sites, S_1, S_2, ..., S_n, interconnected by a computer network as shown in Figure 1. Each site S_i has a local database management system, $DBMS_i$, with a local transaction manager, LTM_i. The MDBS software is distributed among the sites in the form of n software modules. Each module, $MDBS_i$, located at site S_i is interconnected with the other $MDBS_j$ modules by a communications network, but is otherwise independent of any other $MDBS_j$ module. All interaction between the sites takes place via these modules. In particular, the synchronization between the sites is managed by the $MDBS_i$.

All transactions and subtransactions at site S_i are executed by the LTM_i, which is not able to distinguish between transactions and subtransactions. Hence, if no confusion arises, we refer to a subtransaction as well as a local transaction, by the common term "transaction". A subtransaction for a global transaction T_i that executes at site S_j is denoted by T_{ij}.

Each LTM_i is assumed to support the following two types of operations:

1 Introduction

In recent years, there has been has been considerable research effort directed to address the issues of transaction execution in an environment where the data is accessed from a number of different locations that have autonomous concurrency control. The environment consists of a distributed system with a *database management system* (DBMS) present at each location. The practical motivation for such integration arises from areas as diverse as multidatabase transaction management (e.g., see [SKS91]), and distributed —DBMSs (e.g., see [Son88, LL90]). In the former, the need arises in order that pre-existing DBMSs can be integrated into a single, homogeneous, distributed DBMS; whereas in the latter, the autonomous loci of control are necessary in order to ensure that the varying loads in the distributed environment affect a local site minimally. We use the term *multidatabase system* (MDBS) to refer to such an integrated system in order to conform with existing nomenclature.

There are two kinds of transactions that execute in an MDBS system. Transactions that access data located at only a single site are referred to as *local* transactions, whereas those that access data located at several different sites are referred to as *global* transactions. The latter access data by means of a subtransaction executed at each site in question. The problem for transaction management in an MDBS environment, is to identify the conditions that will guarantee that the concurrent execution of both global and local transactions will ensure the database consistency by ensuring globally serializable executions.

The existing results in the area of MDBS transaction management typically assume a single site that coordinates the executions of all the global transactions by the use of a synchronization protocol, and also, that no modifications may be made to the underlying DBMSs (e.g., see [BST90]). The former assumption may cause problems in the event that some sites or communication links fail (e.g., see [SKS91]), and the latter assumption is not necessary in many cases (e.g., in an MDBS used for real-time environments — see [Son88]). Also, the mechanisms that have been suggested thus far only provide sufficient conditions to guarantee globally serializable executions when used in conjunction with a synchronization protocol. Hence, it is unclear as to what extent the interleaving among the operations of the subtransactions and local transactions must actually be affected, and consequently, to what extent the potential concurrency is inhibited.

The research presented in this paper addresses, and resolves, the above deficiencies in maintaining globally serializable executions in an MDBS environment. To begin with, we assume that each site generates *view serializable* (VSR) schedules locally; VSR is the most general class of schedules accepted when only syntactic information about the transactions is available (e.g., see [Pap79, Pap86]). We describe a synchronization protocol that may use distributed coordinator sites

Serializability among Autonomous Transaction Managers

Nandit Soparkar[1*]
Henry F. Korth[2]
Avi Silberschatz[1†]

[1]Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712-1188 USA

[2]Matsushita Information Technology Laboratory
182 Nassau Street, third floor
Princeton, NJ 08542-7072

Abstract

Ensuring the serializability of transaction executions in an environment consisting of several autonomous sites is a current research effort. While several *ad hoc* schemes have been proposed that are sufficient to ensure serializability, it is not clear what conditions are necessary. We explore this question in an attempt to delineate those classes of concurrency control protocols that exist at local sites that may be integrated. Our research indicates the existing or future protocols that are amenable to integration, and also suggests the means to do so. We provide a characterization for integrating the transaction schedules from autonomous sites in a manner similar to that used in centralized concurrency control.

*Work supported by an IBM Graduate Fellowship.

†Work partially supported by NSF grants IRI-9003341 and IRI-9106450, by the Texas Advanced Technology Program under Grant No. ATP-024, and by grants from the IBM corporation and the H-P corporation.

SERIALIZABILITY AMONG AUTONOMOUS TRANSACTION MANAGERS

Nandit Soparkar
Henry F. Korth
Avi Silberschatz

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712-1188

TR-92-49 December 1992

DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN

AUSTIN, TEXAS 78712