CONSISTENCY MANAGEMENT IN DISTRIBUTED DATA BASES: A SELECTIVE ANALYSIS

Mohan L. Ahuja

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

TR-231 May 1983

Table of Contents

1. Introduction

This report develops a classification scheme for consistency management protocols for distributed systems, describes and analyses a number of significant protocols in the context of this classification. The second topic of the report is description and analysis of major papers dealing with performance evaluation of consistency management protocols.

The report has 7 sections, the first being this introduction. The second section defines the terms used in this report and summarizes the problem of consistency in distributed data bases. The third section proposes a classification of the protocols proposed for achieving consistency. The fourth section summarizes some of the many protocols surveyed by this author. It should be mentioned that many protocols not summarized here are also important. We had to make a choice among many, for illustrating the structure of each class defined in Section 3, and so those found to be the best for this purpose were included. The fifth section identifies certain evaluation and system parameters affecting performance of these protocols. Results of studies conducted, [REI79a], [MOL78], and [LIN81], to evaluate the performance are presented in the sixth section. The seventh section defines the need for further research in evaluation of consistency management protocols.

2. The Problem Definition

Before stating the main problem of consistency management in distributed data base systems, we shall define some terms used throughout this report.

Data Base: A Data Base(DB) consists of named data objects called entities. The terms data objects and entities are used interchangeably. By the Data base definition the value of these entities may have to be related in some way which must always be satisfied. These relations are called **Integrity Constraints** . A DB which satisfies the Integrity Constraints is said to be consistent.

Distributed Data Base:
In a Distributed Data Base (DDB) data resides on more than one site connected through some network. Various sets of entities may reside at more then one site, in which case the DDB is said to have data replications. One extreme case could be the entire data replication.

Transaction: A transaction T_i is a finite sequence $\{a_{i,1}...a_{i,n}\}$ of atomic operations. An operation could be a read($r_{i,x}$), write($w_{i,x}$), lock($l_{i,x}$), unlock($u_{i,x}$) or in general any other atomic step in the straight-line program T_i. If a transaction is run by itself on a consistent DB it leads to a consistent state. By induction if a series of transactions S is run on a consistent DB, it transforms to another consistent state. For the purposes of consistency management we can represent a transaction T_i by a series of read($r_{i,j}$) and write($w_{i,j}$) actions, where subscript j is the order of this action in the transaction.

Log: A log L_s is a sequence of atomic actions. A transaction log T_s is a sequence of transactions.

Equivalence: Two transaction logs T_{s1} and T_{s2} are logically equivalent if each of them when executed upon a DB in the same state leads to the same final state.

Serial Execution:
If all the transactions execute strictly one after another the execution log is said to be serial.

Concurrent Execution:
When many users initiate their transactions without being aware of each other, the atomic actions of the transactions may be executed in an interleaved fashion. This interleaved execution of atomic actions of different transactions is referred to as concurrent execution as opposed to the serial execution defined earlier.

Since the transactions are defined to be straight-line programs, even when concurrent execution is permitted, atomic actions of any one of these transactions are still executed in the serial order specified by the transaction.

Serializable Execution:
If an execution log has an equivalent serial execution it is said to be serializable.

Scheduler: A scheduler, as defined in this context, transforms a stream of requests for atomic actions into a schedule that preserves the consistency of the DB.

Certifier: If we have a 'do nothing' scheduler all the steps are scheduled immediately and the resulting changes are kept in temporary copies. At the end of the execution of each transaction a certifier checks whether or not the transaction ran without creating any inconsistency. If the transaction is certified the changes are made to the DB.

Protocol: A protocol in its most general sense, as defined by Ullman, is simply a restriction on the sequence of steps that a transaction may permit. In the case of DDBS it may be thought of as a set of rules followed by each site in the distributed network , which when followed, leads to implicit or explicit communication required to maintain the consistency. For the purpose of brevity we shall use the term protocol to imply protocols for concurrency management. If the set of rules is checked by a scheduler the protocol is said to be **scheduler based**. If after executing a transaction the certifier checks that no rule was violated the protocol is said to be **certifier based**.

Fixed Point Set:

The fixed point set of a scheduler based protocol is the set of all the input sequences, all steps of which would be executed by the scheduler without delay. In certifier based protocols the fixed point set is the set of all the sequences which would be certified to be correct.

The fixed point set is a comprehensive measure of performance of a protocol. By the definition of a fixed point set, it lists all the sequences which will retain the consistency, without adding any delay whatsoever.

The Problem:

A consistent state is one which satisfies the integrity constraints. It may however be impossible to execute a transaction consisting of a sequence of atomic actions without temporarily violating the integrity constraints. Now, if concurrent executions are permitted, one transaction may see an inconsistent state created by another transaction. Interleaved execution may also lead to an unintended state. One example of each-i.e. violation of integrity constraints and creation of an unintended data base state-follows.

Consider a DB for a bank wherein a customer has two accounts: he submits a transaction to transfer money from one account to the other account. This transaction while transferring money from one account to another could create a state in which the money has been subtracted from one account but still has not been added to another. This is so since the operation of subtraction is an atomic action, while the sequence of subtraction and addition is not. The integrity constraint requires that sum of the two balances should be equal at all times during the transaction execution to the sum that existed before, which is being temporarily violated. Meanwhile another transaction reading the sum of the balance of the two accounts may see this inconsistent state.

Another instance, where concurrent execution may lead to an execution log creating an unintended state, is when one transaction T_1 is manipulating entities x & y and at the same time another transaction T_2 is manipulating y. Suppose that T_1 does x=x+1 then y=y+1, while the T_2 does y=y*y. Independent execution of both the transactions would run correctly: but the interleaved execution log x=x+1, y=y*y, y=y+1 leads to an unintended state.

The two illustrations explain the consistency management problem in a simple manner, although very complicated situations may arise. Hence care must be taken while permitting interleaved executions. This is the concurrency control problem, which has been studied extensively in the literature and is the main issue in this report.

Assuming that no semantic information is available about the transactions, it has been proved that an interleaved execution log E will lead to a consistent state if it is serializable. The order of transaction execution in the equivalent serial execution may be any permutation of the set {All the transactions in E}. It is easy to see that this condition also ensures that no transaction will see parts of different DB states. However, some weaker condition will suffice if some semantic information about the transactions is available. In this report, we will assume that no semantic

information is available, hence serializability or serial equivalence is the minimum criterion for preserving consistency.

In DDBS some additional issues arise. One of them is data replication management. If data replication exists, some mechanism to enforce consistency between copies is required. The other problem is that of providing immunity against lost messages. There are other issues, common with operating systems, such as system failure, deadlocks and starvation, which take a different shade in DDBS. Concurrency and consistency management protocols must deal with these issues. In this report, we shall address the central issue along with the data replication management problem, and touch upon the others whenever the need arises.

3. Basic Classification of the Protocols

Numerous protocols have been suggested for maintaining consistency in DDBS, which permit concurrent execution of transactions. Some are applicable to specific network structures, others are more general. Some of them apply to replicated data banks. Before discussing a few protocols of the many surveyed, we shall define a classification of protocols which will be helpful in studying them.

One criterion of classification is whether a protocol is scheduler based or certifier based. There could be some protocols which use a mixed policy in which part of the work is done by the scheduler and part by the certifier. In some cases the scheduler may decide that a transaction needs to be resubmitted. For our further discussion we will presume all the protocols to be schedulers, except when we describe the protocols in Section 4.

There are other ways of classifying families of protocols. We shall discuss one based upon the tool used to achieve one of the most important tasks. The concurrency control problem basically consists of four tasks. The first is to assign an order to all the transactions. The second is identification of conflicting transactions and conflicts. The third is to realize the inter-site synchronization required to achieve this order for the conflicting transactions. The fourth is to achieve the required intra-site synchronization. We shall classify the protocols based upon the **tool** for meeting the inter-site synchronization requirements. There are three basic tools to do so, locks, timestamps and token. Timestamps can be used in two entirely different ways. We propose the following four classes:

1. Locking Protocols.
2. Protocols using timestamped transactions.
3. Protocols that use timestamped entities
   i.e Multiversions of entities.
4. Token passing protocols.

It is possible that a protocol may be classified under one of the four because of the tool used to realize inter-site synchronization. Nevertheless, it may use other tools to perform the other three tasks. For instance, the locking protocols use timestamped transactions to achieve global ordering, which could be realized at each site using locks. The token passing protocols also use timestamped transactions for global ordering. Token passing and timestamp based protocols use locks for intra-site mutual exclusion.

Performance of each of these classes can be comprehensively measured by the fixed point set of all the realizable schedulers/certifiers in that class. Many other parameters could be used to evaluate the performance. These will be discussed in Section 5 of this report. In the remainder of this section, we shall explain what these tools are and how they are used under each class. We shall also classify various protocols without giving any details of the protocols.

3.1 The Locking Protocols:

A DB can be partitioned into entities, which can be locked. By locking an entity a transaction can prevent other transactions from accessing it, until it releases the lock. Various implementations of locks are possible. The larger the entity, the less will be the overhead for locking. But at the same time a larger entity size leads to a lower level of permissible concurrency. Both the implementation of locks and the size of entities are irrelevant to the understanding of the protocols, and will not be discussed further. The level of concurrency can be improved by having various lock modes. These lock modes could be mutually permissible, they may be compatible and/or convertible to one another. By identifying these properties, execution sequences which would otherwise be forbidden could be permitted. This obviously increases the concurrency and size of the fixed point set. Some of the terms related to these modes are defined below to help in understanding the protocols summarized in Section 4.

Definition: A **read mode** lock held by a transaction T on an entity e allows it to read the value of e and use it for computations, as often as it needs, till it releases the lock. We shall denote this mode by R.

Definition: A **write mode** lock held by a transaction T on an entity e allows the transaction to read and/or write and use the value of the entity for computations, as often as it needs, till it releases the lock. We shall denote this lock mode by W.

Definition: A locking mode X is said to be **compatible** with a mode Y if a transaction may acquire X on an entity while another transaction holds Y on it. For example a R lock an entity is compatible with another R lock on the same entity but is incompatible with a W lock on this entity.

Note that the compatibility relationship is asymmetric i.e. it may be possible that X be compatible with Y but the Y may not is compatible with X.

The locking protocols use locks for the inter-site mutual exclusion in case of conflicting transactions. Realizing locks in a distributed system is very expensive in terms of the overhead involved. To reduce such overhead locks could be placed only on a specific site, i.e. a primary site, or on a specific copy for each entity i.e a primary copy.

Unless the logical data base is specified as having a specific structure (e.g. Directed Graph Structure) all the transactions must follow two phase locking protocol to ensure serializability. A

two-phase locking protocol simply specifies that in each transaction all the locking operations must precede any unlocking operation. Thus we have the following two subclasses under the locking protocols.

1.Two-phase locking.

2.Non-two-phase locking

3.1.1 Two-phase locking protocols

In the two-phase locking it is easy to see that deadlocks are possible. Consider the following sequence of operations by two transactions T_1 and T_2, listed in order of their execution,

T_1:Lock A (granted)	
	T_2:Lock B (granted)
T_1:Lock C (delayed till T_2 releases it)	
	T_2:Lock B (delayed till T_1 releases it)

Here T_1 is waiting on T_2, T_2 is waiting on T_1, and they will keep doing so, and deadlock results. To avoid deadlocks we could set an order to all the entities and stipulate that all the transactions request locks only in the said order. Alternatively, when a transaction has been permitted to start executing, it may put intention locks on all the entities it would ever need. These locks may be used to rule out the possibility of a deadlock before permitting any other transaction to put its intention locks. Thus we need to superimpose a mechanism on top of the two-phase locking protocol to ensure deadlock freedom.

As already mentioned there are two ways of achieving centralized locks. In the first case, one site is designated to be primary and transactions running on all sites seek locks from it. It is clear that the primary site protocols would have high communication requirements and that the primary site would tend to be a bottle-neck. [REI79a] and [MOL78] evaluate performance of two somewhat different primary site protocols and derived strikingly different results. The behavior is further discussed in Section 6.

In the second case, one copy of each entity is designated to be primary and locks are sought only on these copies. Here communication overhead tends to distribute over all the sites, since different entities have their primary copy on different sites. An interesting variation of a primary copy is 'a migrating primary copy' proposed by [MIN79].

It can be seen that inter-copy inconsistencies are transparent to the transactions. However, since

the read only transactions may be permitted to read from the non-primary site/copy, we need another superimposed mechanism.

3.1.2 Non-two-phase locking

As the name suggests these protocols do not use the two-phase locking policy. Very few protocols have been suggested which belong to this class. One of these, proposed in [SIL81] is applicable for hierarchically organized DB systems. A summary of this protocol is given in section 4.

3.2 Protocols using timestamped transactions

The concurrency control problem essentially that of achieving a total ordering between events. An event could be the transaction initiation time at various sites. These sites have different clocks. As already explained the concurrency control involves four steps. First ordering is assigned to all the transactions. Having assigned the ordering, an analysis needs to be done to identify conflicts. The worst case may require a serial execution of transactions. Then the scheduler implements the ordering in two steps i.e inter-site and intra-site synchronization. In all the protocols timestamps are used to globally order the transactions. **For this class of protocols timestamps are also used to mutually exclude the conflicting operations initiated by the different sites.** The actual communication sequences of different protocols in this class may be different and so these sequences are discussed under specific protocols in Section 4.

Two schemes [REE78] and [LAMP78] have been proposed for generating system wide the unique timestamps. In [REE78] all sites are assigned their site numbers, when an event occurs each site reads its local time. The timestamp of each event is a tuple (local time, site number). For events with identical local times their site numbers are used to break the tie.

In [LAMP78] all the clocks are event driven and the tie is broken as in [REE78]. Let us take the diagrammatic representation of the problem as given in figure-1. The parallel lines represent the time scales of different sites, in terms of the events. Each site j has an event clock E_j. All these event clocks are initialized to a constant. They monotonically increase with the occurance of events and may not be synchronized all the time. In our system there could be two type of events, initiation of a transaction and receipt of a message from another site: these will be explained shortly. When a transaction T_i is initiated at site j, E_j is incremented , T_i is assigned a timestamp $ts_{(j,i)}$, and a message $M[(ts_{(j,i)}),j]$ is sent to all the other sites. Message transmission is represented in the figure by an arrow directed towards the receiving site.

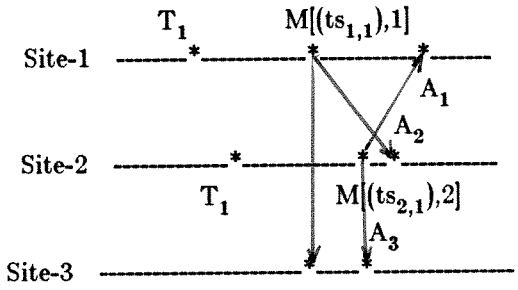

figure-1

When a site k receives a message $M[(ts_{(j,i)}),j]$, it sets its clock to the maximum of the E_k and the $[ts_{(j,i)}+1]$. This essentially is equivalent to tilting the directed edges just enough to make the angle A_k a little more then a right angle. This ensures that no messages are received without establishing the precedence of a send message event to the corresponding receive message event. This gives us a total ordering for all the events except those carrying the timestamp of the same value. However if each site is assigned a priority, say in the same order as its site number and this number is tagged to each message the site sends, then the receiving site can independently break the tie based on the site priority. Thus we get a total ordering.

3.3 Protocols that use multiversions of the entities

In this class of protocols the transactions are assigned a timestamp , as explained in 3.2. The protocol proposed by [REE78] is presented here to explain the basic functioning of this class. Each entity has a field which contains the timestamp of the transaction which created it and also a field which contains the timestamp of the transaction which read it last. If a transaction T_i is trying to create a new version of an entity which has a 'last read ' timestamp greater then ts_i, it will be rejected, since letting it write would mean going back in time. If an entity is to be rewritten a new version is created while the earlier ones are retained. Hence when a version is being written earlier ones are available for read operation by other transactions. This leads to more concurrency. Since all the entities created by a transaction are committed simultaneously, it ensures that all the operations in a transaction see the same state of the data base: **thus the inter-site synchronization requirement is practically obviated.** These protocols permit a very high level of concurrency. It has a disadvantage of requiring many copies of each entity: hence a large memory is required. However, if laser technology is used we shall get non-erasable copies and this disadvantage will no longer be a disadvantage. These protocols, by virtue of the multiversions of an entity, are highly robust to system failures. In practice only a finite number

of old versions need to be retained. The representative protocols of this class are ones suggested by Reed [REE78], Bayer [BAY80a], and Stearn & Rosenkvatzs [ROS77]: of these, Reed's protocol will be summarized in Section 4.

3.4 Token passing protocols

These protocols require that there be an order for passing the token. Permission to access the entities, requiring mutual exclusion between sites, is passed from one site to another in a predetermined order. A token is used as a symbol of the permission. **Since there is only one token, inter-site mutual exclusion is ensured.**

A system state table is also passed on along with the token. It contains all the information required to ensure consistency. A detailed account of the information contents of the state table is given in Section 4, while discussing the protocol proposed by [GRE81]. Here it would suffice to say that the timestamps of the transactions and intention locks are the key to convey the required information. These protocols have a high throughput achieved at the cost of a higher turnaround time. The turnaround time will be high even if the transaction requires little or no processing. The performance of these protocols deteriorates with reduction in the load level. This is so since the token may have to be passed to many sites not requiring it before it reaches the site requiring it.

4. Summary of a Few Representative Protocols:

In this section we shall summarize some of the representative protocols of each class.

4.1 Locking protocols

As already said these protocols could be further classified depending upon whether or not they use two-phase locking policy.

4.1.1 The two-phase locking protocols.

Under this class we shall cover only the primary site based protocols because the mechanisms used by the primary copy based protocols are essentially the same. The protocols evaluated by Reis [REI79a] and Molina [MOL78] are discussed here.

4.1.1.1 [REI79a]

Details of this protocol can be found in [REI79a]. Reis basically addresses the problem of evaluating performance of protocols, specifically merits of the centralized i.e the primary site protocols, and the distributed algorithms. The gist of the two primary site protocols evaluated by [REI79a] are given here. In both, a fixed ordering is placed on all the sites. All the locks are exclusive i.e no distinction is made between read and write locks. Locks for a transaction are granted in a fixed order of the locking granules within a site. Again for each transaction all the locks on one site are granted before granting locks on another. This is done in a predetermined order among all the sites. If locks required by a transaction T_i on a site are already held by another transaction, the T_i would wait for release of these locks. When the locks for all the sites have been granted the primary site would then send a 'locks granted' message to the transaction initiation site. The fixed order among sites is used to prevent deadlocks. The second protocol is just like the first, except if the locks required by a transaction T_i for a given site are held by another transaction, all the locks granted for all the lower numbered sites would be released. When the locks are available the acquisition is restarted. Here no transaction waits while it holds any locks.

4.1.1.2 [MOL78]

A summary of the protocol proposed in [MOL78] follows. The protocol presumes a fully replicated copy of the data base on each site. It goes as follows: a site S requests from the primary site all the locks it needs for the transaction T_i. The primary site locally checks for all the locks and, when they can be granted, a message is sent to S. It also assigns a sequence

number to all the updates it grants. Then S goes ahead and executes the transaction. The update generated is sent to all the sites along with its sequence number. The site S makes the updates in its copy of the DB. When the other sites receive an update message they perform the update. The primary site, in addition to performing the updates, releases the locks. The sites keep the sequence number of the latest update performed and delay updates which are out of order.

4.1.2 The non-two-phase locking protocols.

We choose the first proposed protocol in this class of protocols. It is due to Silberschatz and Kedem [SIL80]. It presumes a hierarchically organized data base as shown in figure-2.

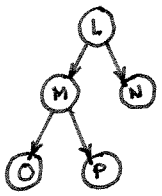

figure-2

Unlike the two-phase locking protocols it is free from deadlocks. It locks an entity, i.e a node in the tree, without implying any locks on its descendant in the tree. A transaction following this protocol must satisfy the following conditions,

1. T_i may lock any node to start-with.
2. T_i may lock other nodes only if it already holds its father in the data base.
3. After unlocking a node it may not lock it again.
4. It may access only those nodes on which it holds a lock.

Note that the transaction need not be two phase. It has been proved that the schedules produced by this protocol are serializable. An intuitive understanding of this fact is easy to see by the following argument: each transaction has a frontier of lowest nodes in the tree on which it holds the locks. The protocol guarantees (conditions 2 and 3) that these frontiers do not overlap. If the frontier of T_i begins above the frontier of T_j , it will remain so, and every item to be locked by both will be locked by T_j first.

4.2 Timestamped Transaction based protocols.

A large number of protocols have been suggested in this class. We shall choose the two representative protocols, one due to Badal [BAD78] and the other titled SDD1 i.e the System for Distributed Data Bases[BER80c].

4.2.0.1 [BAD78]

The protocol presented here is due to Badal, D. Z., and Popeck, G. J., see [BAD78] for details. For each write operation on an object A its new value is bound to a local object. Whenever the next read is scheduled on A, all the other sites supply the local objects, if they have any, which need to be bound to A in the global data base. Thus local objects are bound to the global data base at the time of executing a read operation, hence this protocol is termed as 'read driven' protocol. However, this read driven protocol makes the system failure recovery more difficult. The authors have proposed several means for various degrees of recovery.

The protocol works as follows: each site has a directory giving a list of the objects on each site. When a transaction is initiated at a site, all the entities to be read and written are identified. All the sites involved in read and write operations are then determined and are termed as read and write sites. The transaction initiation site sends a set up message, henceforth to be abbreviated as SUM, to all the read and write sites and assigns a timestamp to the transaction. The SUM contains the definition of the transaction including its timestamp, read and write sites involved, and entities to be accessed. All the sites acknowledge the SUM and a preferred read site is selected. The preferred read site then sends a request to each site asking for updates generated, if any, for each of the entities to be read. All the sites either send the updates or a null message to the preferred read site. The preferred read site then sends read commands to all the read sites along with the updates to be done and SUMs not yet acknowledged. Each site performs the update operations on the entity in the order of their timestamps and then does the read operation. Each read site also prepares write messages and keeps them with itself until the next request message by some preferred site demands these write messages. This protocol is remarkable by virtue of its read-driven nature.

4.2.0.2 [BER80c]

The following protocol is titled 'The System for Distributed Databases 1', abbreviated as SDD1. It has been jointly suggested by Bernstein, P. A., Shipman, D. W. and Rothnie, J. B., Jr [BER80c]. The SDD-1 suggests reduction of synchronization overheads by using pre-analysis of the transactions with-respect-to other possible transaction classes. The system is presumed to be architectured to have Transaction Modules (TMs) and Data Modules (DMs). TMs supervise user

transactions and simulate a non-redundant non-distributed Data Base (DB) for him. DMs are abstract data types which perform their local scheduling. It also presumes a reliable network (RELNET) for the network communication.

At System Creation time certain classes are defined, using simple predicates, in terms of their read and write sets and are assigned to run at specific TMs. This pre-execution classification covers most of the anticipated transactions. To cover all possible types of transactions a global class (having all records in the DB as its read and write set) is defined. The protocol that each transaction class should follow with-respect-to the other classes is determined statically by analyzing the Conflict Graph (CG) of the system. At the time of initiation each transaction T_i is assigned a time stamp ts_i and a class I based as its read set (ri) and write set (wi) . Now this transaction follows protocols with-respect-to other classes as specified by the CG analysis for its class. If transaction can fit in more than one class, it is assigned to the class requiring minimal synchronization.

SDD-1 achieves higher concurrency by using the Write Message rule. The Write-Message rule says, a data item is updated by a write message if and only if the data item's creation time is less than the write message's time stamp. Otherwise it is not and the write is considered to be dead, since it will not be seen by any other transaction.

The CG mentioned above has two nodes (rI, wI) for each transaction class I. It has vertical edges <rI, wI> for all I, horizontal edges <wI, wJ> and diagonal edges <rI, wJ> , <wI, rJ> for each pair (I, J) having some common data involved in the paired nodes of each of these edges. Henceforth we shall denote a read and write operation on a DM x by a transaction T_i by $R_{i,x}$ and $W_{i,x}$ respectively and the timestamp of T_i by ts_i.

Bernstein, et al define four protocols, P1, P2, P3, P4, catering in that order to higher synchronization requirements. P1 ensures that for any i1, i2 in I and j1, j2 in J, whenever $W_{j1,a}$ precedes and conflicts with $R_{i1,a}$, at DM a, and $R_{i2,b}$ precedes and conflicts with $W_{j2,b}$ at DM b, and either $ts_{i1} < ts_{i2}$ or i1=i21 then $ts_{j1} < ts_{j2}$. If $R_{i,a}$ precedes and conflicts with $W_{j,a}$, then P2 ensures that if $ts_k > ts_j$ then $R_{i,b}$ precedes $W_{k,b}$ for every b where both appear and conflict and if $ts_j > ts_k$ then $R_{i,b}$ follows $W_{k,b}$ at every DM b, where they both appear and conflict. P3 is a stronger version of p2 and says that if $R_{i,a}$ conflicts with a $W_{j,a}$ then they must execute in order of their timestamps. P4 is the most restrictive protocol and dictates that the transactions be executed serially in the order of their timestamps.

CG analysis is done as follows. Transactions belonging to classes not on any diagonal edge do not need any synchronization. For conflicting diagonal edges <rI, wK>, <wJ, rI> on a cycle such that only read node of I is involved in the cycle, i follows P_2 with-respect-to j in J and K in K. For two transactions belonging to two classes connected by a diagonal edge in a cycle, say <rI, wJ>, i in I follows P_3 with-respect-to j in J. For transaction i not belonging to any other class but the global class protocol P_4 is followed.

The protocols are implemented by attaching a read condition <ts, {J1 . . . Jn}>. Here ts is a timestamp and J1 ... Jn are the classes of transactions. The read condition is considered to be satisfied when all write messages originated before ts from each class in J1 ... Jn have been executed and no more from the classes have been executed. Selection of ts and the classes depends upon the protocol being used. Write pipelining ensures execution of write messages from each class in their time stamp order. To avoid starvation the NULLWRITE and SENDNULL mechanisms are used.

At the runtime, the Concurrency Monitor at each DM executes as dictated by the read conditions. We would like to close this summary with the mention of the following salient points about the protocol:

1. The architecture makes a strong separation between concurrency control (CC) issues and those of query processing and reliability. One of its advantages is modifiability of the software and its tunability.

2. Even if two transaction classes require synchronization relative to certain data, other classes can concurrently access the data: in fact, other classes can be synchronized for this very data independent of these two classes.

3. The maximum burden of having to provide for a global class is borne by the transaction belonging to it by having them follow P4. Since they occur rarely, it seems to be the right approach.

4. Value to local object binding is done during the execution phase which follows the read phase. At the time of each read operation on an entity all pending local to global bindings of the entity to be read are done. Write messages from classes specified in the read condition must be executed

anytime before the read is executed. This is also a 'read driven' protocol. In the Read driven model of Badal, write messages are attached with the read message enhancing reliability, here Relnet ensures reliability. The recovery mechanism is basically Relnet and global clock based.

5. Whenever the load on the system is high, the performance will degrade. The variance of waiting times before a transaction gets service will be very high, as confirmed by [LIN81].

4.3 Protocols that use multiversions of the entities.

The protocol proposed by Reed is presented next. Refer to [REE78] for details. This protocol has the very important goal of maximizing the site autonomy i.e maintenance of the individual data object should be the responsibility of the site containing it. Coordination of the transaction accessing the same object is done as part of the accessing operations, and locally to the site containing the object. Here many versions of a data object are kept. Each object has an object header which points to the latest version of the object. Each version has a pointer to the next version (in fact last in chronology), start field and end field. The start field contains the timestamp of the transaction that created this version, and the end field contains the timestamp of the transaction that read it last. Initially, the start and the end fields are identical. A read operation reads the latest version written before this read. It also updates the end field. A version is not available to a read operation till it has been committed a transaction has run to a successful completion its possibilities are confirmed and made part of the data base. Note that a read operation may cause rejection of a write operation and hence the transaction, if the write arrives out of sequence. We have summarized just the mechanism for inter-transaction synchronization. There is much more in this protocol which is unique.

4.4 Token passing protocols

One representative protocol of this class is presented next ,refer to [GRE81] for details. The protocol proposes a concurrency control mechanism for a DDB constituted of a ring structured network. The protocol is named "Cooperative Multi-thread Algorithm (CMT)", "cooperative" since control of the DDB passes in a predetermined order for a specific quantum of time and "multi-thread" because each site processes the request in a multi-thread mode. The algorithm suggested makes three assumptions. First, each site should be kept busy by a pool of waiting requests. Second, inter-site communication is based on a logical ring network structure. Lastly the DDB is fully replicated-i.e. all the data resides on each site.

The algorithm performs two functions: cooperative control migration and DDB state migration.

Two tables, a control table and an update table, maintained at each site provide the information needed by the algorithm. The control table specifies the control cycle by a set of tuples giving identity numbers of each site, its predecessor and successor. This table is modified in case of contingencies. The update table contains the current state of the DDB. It has a sublist of locked granules-i.e. data objects-and a modification sublist. This table is passed with the control token. Each site repeatedly performs four steps: waiting, accounting, user and termination. During waiting it just waits for the arrival of the control token.

Accounting commences with the arrival of the token. When a site receives the token it does the following

1. Adds the writing list generated while the token was with other sites.
2. Initializes the subcycle number of these writes to zero.
3. Performs write operations specified and decrements their subcycle number by one.
4. Puts the intention write locks the read locks to be notified to the other sites.

Since the site originating a write initializes the writes subcycle to zero, when it becomes equal to the number of sites in the DDB, it has been notified to all the sites and so can be removed from the state table.

The user step involves reading. To avoid starvation of a site by its predecessor, an enhanced allocation scheme may be required. Such schemes could be easily worked out, although for small granule size, starvation rarely occurs.

The termination step commences at each quantum expiration. It involves migration of control to the successor, along with the update table. After this the site can execute the user process, and generates a new write list, which will be added to the update table when the control token arrives again. If the processing takes longer than a cycle-time, locks will be renewed.

Note that the control table size (3n, where n is the number of sites in the DDB) remains fixed as long as the number of sites remains fixed. The update table size, however, is more volatile. It remains stable for a fixed transaction arrival rate and time quantum. Each lock and each modification is circulated (n-1) times resulting in a minimum of 2*(n-1) messages per update. A lock recirculation would lead to a higher number of messages.

Greene further proposes a node architecture having a front-end processor for communication processing, a multi-processor for application processing and a back-end processor for data management processing. He also suggests tuning of bandwidth of the communication channel, depending upon the size of the update table.

The measure of mutual consistency-i.e. number of granules of one replication which differ with counterparts in all the replications- will always be less than one (unless all activities cease for one cycle). The algorithm is deadlock free, since a claim on all the required granules is required prior to its execution. A failed site can be bypassed by modifying the control table. The problem of recovery is not addressed. It ensures equal access opportunities at site level rather than user process level. Allocation of incremental quantum is under local control, which provides another tuning parameter.

5. Parameter Identification for Performance Evaluation of the Protocols.

The parameters can be classified as system and evaluation parameters. Those specifying assumptions of a protocol or the mechanism that it uses would be classified as the system parameters. Evaluation parameters are those which could be used as a metric to evaluate the performance. System parameters are important because the proposed protocols are so diverse that it may not be meaningful to judge them as a whole. However, if the dependency of the evaluation parameters to the system parameters is established a better understanding could be gained. Some of these parameters under each class are given below.

A. System Parameters.

1. Extent of preanalysis of transactions.

2. The class, as defined in Section 4, to which the protocol belongs.

3. Degree of centralization.

4. Assumptions about the network topology.

5. Time of data object to data base binding.

6. Whether or not obsolete information is read.

7. Work load parameters.

7.1 Degree of interference.

7.1.1 Average number of transaction conflicts.

7.1.2 Average number of transactions in a conflict.

7.1.3 Average time between conflicts.

7.1.4 Average duration of conflicts.

7.1.5 Average number of data objects involved in a conflict.

7.2 Degree of locality of transactions.

This could be measured either as local v/s non-local or by

$$DL = \frac{\text{Average \# of sites accessed by a transaction}}{\text{Total \# of sites in the network}}$$

7.3 Transaction arrival rate.

7.4 Average I/O requirement of a transaction.

8. Locking granuality.

9. Network communication speed or Slow v/s Fast networks.

10. Number of nodes in the network.

B. Evaluation parameters.

1. Fixed point set.

2. Transaction rejection rate.

3. A measure of delay introduced by concurrency.

4. Number of messages.

5. Extent of recovery, from system failure, inbuilt in the concurrency control algorithm.

6. Deadlock freedom.

7. CPU utilization.

8. I/O utilization.

9. Average response time.

6. Salient Results of the Performance Evaluation Work Done So Far.

We shall present the results of three papers-[REIS79a], [MOL78], [BAD80b]- in conjunction with each other and then summarize the results of [LIN81]. Reis [REI79a] and Molina [MOL78] derive seemingly contradictory results about centralized or Primary Site based protocols, henceforth referred to as PS, and Distributed Algorithm, henceforth referred to as DA. Molina concludes that the centralized locking algorithms perform considerably better then DAs except in case of very high I/O activities, while Reis concludes that DA does better. Badal [BAD78] provides a plausible explanation of this contradiction. He points out that these results are applicable for different domains of the system parameters, namely communication speed of the network, locality of transactions and their arrival rate. Thus these results are complimentary to each other, as will be clear from the following.

Reis considers four system parameters. These are type of transactions, % of non-local transactions, locking granuality and speed of the communication channels in the network i.e 7,7.2,8,9, respectively of the system parameters in Section 4. The evaluation parameters that he considers are CPU utilization, I/O utilization, and average response time: these are 7,8,9 respectively of the evaluation parameters listed above.

He finds that when most of the transactions are non-local and the network is slow then the PS performs better then the DA. While if the most of the transactions are non-local and the network is fast the DA does better. Intuitive feeling would also suggests that if the communication time is small and the PS is followed the primary site tends to bottleneck. The PS also performs better when most transactions are non-local and require a fixed pattern of locks and require to lock a large number of granules.

When most transactions are local and the network is fast both do equally well, while if most transactions are local and the network is slow the DA does better.

Molina considers the following system parameters: number of sites in the network (A.10), interarrival time between transactions (A.7.5), mean base set size (A.7.1.5), average I/O requirements of transactions (A.7.4), and speed of the network (A.9). Among the evaluation parameters in addition to the Reis's list he considers number of messages. Note that Molina's set

of parameters is a superset of Reis's set, except for the locking granuality which Molina does not consider. All the transactions here are non-local since each has at least one update operation.

Molina finds that for a low to moderate arrival rate the PS is better, while for a high arrival rate the DA does better. He also finds that for a high probability of transaction conflicts the PS does better while for low probability the DA outperforms the PS. Also for high I/O activities the DA does better since I/O operations do not become a bottleneck.

Lin [LIN81] compares performance of SDD1 and Dynamic Time Stamping Method (DTM). The DTM is an extension of SDD1 wherein the timestamp of a transaction is bumped to avoid certain rejections. Also DTM does not tag to a Data Item the time stamp of the transaction that updates the data item, so no dead-writes are permitted. The SDD1 presented in [BER80c] is different from its actual implementation which has been used by Lin. He proposes a little more transaction analysis-i.e.CG graph analysis- which helps in identifying cases where bumping of timestamps would obviate rejections.

Though the comparison is valid only for a very narrow range of transactions, the results are appealing. Average response time, abbreviated as ART, and standard deviation of response time have been used as measure functions. High dependence of performance of one class, when P3 of SDD1 needs to be followed, on arrival rates of transactions in the other classes is strongly established. Contradictory behavior of rejection rate and ART with varying arrival rate of the transactions in the controlling class is demonstrated. Utility of NULLWRITES for reducing dependence of ART is confirmed.

7. Conclusions and Summary.

A large number of protocols have been proposed but little work has been done for evaluating their performance. We feel that the first step for evaluating these protocols is to identity all the system and evaluation parameters both as predicates and as metrics. The next step would be to isolate the one-to-one dependency of the evaluation parameters on the system parameters and their correlation. A sensitivity analysis may also be conducted. Lastly the protocols may be evaluated, and complex dependencies may be inferred from these results. This report presented a classification for studying these protocols and summarized a few protocols. Furthermore it identified a quite exhaustive list of evaluation and system parameters. Lastly it summarized the results of four performance evaluation studies.

Bibliography

[BAD78] Badal, D. Z, Popeck, G. J., "Proposal for Distributed Concurrency Control for Partially Redundant Distributed Database Systems", Proc. 3rd Berkeley Workshop on Dist. Data Management and Computer Networks, p. 273-288 (1978).

[BAD80b] Badal, D. Z., "The Analysis of the Effects of Concurrency Control in Distributed Database System Performance", Proc. of the VLDB, p. 376-383 (1980).

[BAY80a] Bayer, R., Elhardt, Klaus, Heller, H. and Reiser, R., "Distributed Concurrency Control in Database Systems", Proc. VLDB, pp. 275-284 (1980).

[BAY80b] Bayer, R., Heller, H. and Reiser, A., "Parallelism and Recovery in Database Systems", ACM TODS 5, 139-156 (1980).

[BER78] Bernstein, P. A., Rothnie, J. D., Goodman, N. and Papadimitriou, C. A., "The Concurrency Control Mechanism of SDD-1: A System for Distributed Databases (the Fully Redundant Case)", IEEE Trans. on Software Engineering, p. 154-168 (1978).

[BER79] Bernstein, P. A., Shipman, D. W., and Wong, W. S., "Formal Aspects of Serializability in Database Concurrency Control", IEEE Trans. on Software Engineering, Vol. 5, p. 203-215 (1979).

[BER80c] Bernstein, P. A., Shipman, D. W. and Rothnie, J. B., Jr., "Concurrency Control in a System for Distributed Databases (SDD-1)", ACM Trans. on Database Systems, 5, 18-51 (1980).

[BER80d] Bernstein, P. A. and Goodman, N., "Time Stamp Based Algorithms for Concurrency Control in Distributed Database Systems", Proc. VLDB pp. 285-300 (1980).

[GRE81] Greene, Richard J., "An Alternative Approach to Distributed Database Updating", Proc. NCC 50, 481-485 (1981).

[KED80] Kedem, Z. and Silberschatz, A., "Non Two-Phase Locking Protocols with Shared and Exclusive Locks", Proc. VLDB, pp. 309-317 (1980).

[KUN79] Kung, H. T. and Papadimitriou, C. H., "An Optimality Theory of Concurrency Control for Databases", Proc. ACM SIGMOD 79, p. 116-126.

[KUN81] Kung, H. T. and Robinson, J. T., "Optimistic Methods for Concurrency Control", ACM Trans. on Database Systems, 6, 213-226 (1981).

[LAMP78] Lamport L., 'Time, Clock and the ordering of events in a distributed system', CACM, July 78, Vol.21,#7.

[LAM79] Lampson, B. W. and Sturgis, H. E., "Crash Recovery in a Distributed Data Storage System", Technical Report, Xerox Parc, April 1979.

[LEL78] Le Lann, G., "Algorithms for Distributed Data Sharing System Which uses Tickets", Proc. 3rd Berkeley Workshop on Dist. Data Management and Computer Networks, p. 259-272 (1978).

[LIN79] Lin, W. K., "Concurrency Control in a Multiple Copy Distributed Database System", Proc. 4th Berkeley Workshop on Distributed Data Management and Computer Networks, p. 207-220 (1979).

[LIN81] Lin, W. K., "Performance Evaluation of Two Concurrency Control Mechanisms in a Distributed Database System", Proc. 1981 ACM SIGMOD Conf., p. 84-92 (1981).

[MEN78a] Menasce, D. A. and Muntz, R. R., "Locking and Deadlock Detection in Distributed Data Bases", Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, p. 215-234 (1978).

[MIN79] Minoura, T., "A New Concurrency Control Algorithm for Distributed Database System", Proc. 4th Berkeley Workshop on Distributed Data Management and Computer Networks, p. 221-236 (1979).

[MOL78] Garcia-Molina, Hector, "A Performance Comparison of Two Update Algorithms for Distributed Databases", Proc. 3rd Berkeley Workshop on Distributed Data Management and Computer Networks, p. 108-122.

[REE78] Reed, D. P., "Naming and Synchronization in Decentralized Computer Systems", Technical Report MIT/LCS/TR205, September 1978.

[REE79] Reed, D. P., "Implementing Atomic Actions on Decentralized Data", Proc. of 7th Symp. on Operating Systems Principles, pp. 163-175 (December 1979).

[REI79a] Reis, D. R., "The effects of Concurrency Control on the Performance of Distributed Data Management Systems", Proc. 4th Berkeley Workshop on Dist. Data Management and Computer Networks, p.75-112 (1979).

[ROS77] Rosenkrantz, D. J., Sterns, R. E. and Lewis, P. M., "A System for Concurrency Control for Database Systems", Proc. 2nd Berkeley Workshop on Distributed Data Management Computer Networks, p. 132-145 (May 1977).

[ROS78] Rosenkrantz, D. J., Sterns, R. E., and Lewis, B. M., "System Level Concurrency Control for Distributed Database Systems", ACM Trans. on Database Systems, Vol. 3, p. 178-198 (1978).

[SIL80] Silberschatz, A. and Kedem, Z., "Consistency in Hierarchical Database Systems", JACM, p. 72-80 (1980).

[SKE81b] Skeen, Dale, "Non-Blocking Commit Protocols", Proc. ACM-SIGMOD International Conference on Management of Data, pp. 133-142 (1981).

[STE81] Stearns, R. E. and Rosentrance, D. J., "Distributed Database Concurrency Controls Using Before Values", Proc. 1981 ACM SIGMOD Conf., p. 74-83 (1981).

[STO79] Stonebraker, M., "Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES", IEEE Transactions on Software Engineering, 5, p. 188-194 (1979).

[THO79] Thomas, R. H., "A Majority Consensus Approach to Concurrency Control", ACM Trans. on Database Management, Vol. 4, p. 180-211 (1979).