DISTRIBUTED DATA BASE MANAGEMENT
SOME THOUGHTS AND ANALYSES*

C. Mohan

May 1979 TR-129

Department of Computer Science
The University of Texas at Austin
Austin, Texas 78712

---

*This work was supported in part by the Air Force Office of Scientific Research under grant AFOSR77-3409.

DISTRIBUTED DATA BASE MANAGEMENT
SOME THOUGHTS AND ANALYSES

C. Mohan

Software and Data Base Engineering Group
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

21 May 1979
Technical Report TR-129

1.0 INTRODUCTION

With the ever increasing popularity of distributed processing, the area of distributed data bases (DDB) has become a 'hot' topic for research. But the work in this area has been very scattered and individualistic. Only recently a concerted group effort has been made to identify some important issues and problems [Bachman 1978a, CODASYL 1978]. These efforts are still in an embryonic stage. Much more work needs to be done in this direction. The works of the ANSI/X3/SPARC Study Group on Distributed Systems, the International Telegraph and Telephone Consultative Committee (CCITT) Study Group VII, and the ISO Sub-Committee (ISO/TC97/SC16) on Open System Interconnection [ANSI 1978, Bachman 1978b, ISO 1978, MacDonald 1978] are also relevant in this context. As pointed out in [Adiba 1978] decentralization of data processing could very well lead to anarchy unless carefully controlled by adequate communication and co-operation protocols.

A coherent set of principles and methodologies needs to be developed for the design, implementation, administration, maintenance and modification of distributed data base management systems (DDBS) and distributed data bases (DDB). While some technical problems have been attacked and solved, almost all the managerial and organizational problems are yet to be even attacked. The developments in this area have been of a highly evolutionary nature rather than revolutionary. One cannot accept the assertion, made in [Schneider 1978], that from a technical point of view the difference between centralized data base systems and distributed data base systems is not very big. If this assertion were to be true we would be having many working DDBS by now (see [Rothnie 1978]). Because of the fact that,

---

This work was supported in part by the Air Force Office of Scientific Research under grant AFOSR77-3409.

in the DDB environment, not only the data but also the control and status information about which process is accessing or modifying what information is also distributed, the solutions to deadlock, concurrency control and resiliency problems would turn out to be much more complex than in the centralized environment. In addition to these there are the data translation, operating system coordination, efficient DDB query decomposition (into optimal local queries) and (possibly) distributed directory maintenance problems, which are unique and very specific to the former environment. Le Lann has presented, in a formal manner, some of the characteristics of distributed systems and has shown how they are not simple extensions of centralized systems [Le Lann 1977]. The existence of non-deterministic time references and unpredictable propagation delays has been shown to make it impossible to maintain the global state of a distributed system.

As has been pointed out in [Adiba 1978] DDB are the converging point of networking and data base technologies. Hence those aspiring to do research in the DDB area should undertake an in-depth study of the developments (particularly with respect to communication and co-operation protocols and their verification, performance evaluation, error detection and recovery, resource management, optimization of design for various metrics and so on) in the computer networks area. Research in the areas of operating systems [Mohan 1978a], data base management [Mohan 1978b] and computer networks [INFOTECH 1977 1978] should be closely followed and interrelated. Unless that is done lot of effort will be expended in duplicating some work that has already been done.

In the following sections I have presented some of my observations and opinions on the nature of DDB research that has so far been performed and some of the shortcomings of those research activities. The aim of the analysis has been to be very objective, pragmatic and critical, and to point to some needed work. The desire has been to provide a perspective different from those of some existing survey papers [Adiba 1978, Maryanski 1978a, Rothnie 1977, Severino 1977]. In addition to very specific issues I have looked into some global conceptual issues also. My observations and analyses of some proposed DDBS designs and multilevel architectures for DDBS can be found in [Mohan 1979a 1979b]. Some of my ideas on DDB design and administration can be found in [Mohan 1979b 1979d].

2.0 ANALYSES OF PROPOSED MECHANISMS

One of the unfortunate aspects of research in the DDB area has been that many researchers have been studying and proposing "new and/or better" solutions for certain problems more or less in isolation from other related problems. The consequence of this situation is that the impact of the proposed solutions on the unconsidered problems are neither brought out in the papers proposing the solutions nor are they apparent to the readers. The mechanisms of DDBS are so much intertwined that changes proposed even for small parts of the system are highly likely to have repercussions throughout the system (for e.g. changes in system recovery and transaction backup mechanisms' requirements would affect the locking mechanisms). Some of the tradeoffs made are not likely to be apparent. My argument is that a comprehensive analysis of a proposed mechanism must be done before one could accept the claim that the proposed mechanism is better than existing mechanisms/solutions.

2.1 Integrity And Security Constraints

Most authors claim that one of the advantages of DDBS is the increased security that it affords to data stored at each of the nodes. While the potential for this seems to exist one is yet to see how this is achieved. Much work needs to be done in devising mechanisms for the retention and enforcement of integrity and security constraints. The different choices available for the storage of these constraints need to be studied. This would be impacted by the level of network transparency that is in force. Depending on the application environment updates of the constraints would be frequent or infrequent. A study of the environment would influence the choice of strategies to be employed. It would be a non-trivial task to decide on how to distribute these mechanisms in the network (i.e. where would the enforcement mechanisms reside?). Except for some initial work on access authorization reported in [Chang 1977] and on integrity constraints reported in [Badal 1978a, Stonebraker 1977] practically nothing has been published on these topics.

A proposal for the enforcement of semantic integrity (SI) constraints has been put forth in [Badal 1978a]. This requires having, in addition to the data in the DB, what is termed as SI data, which is part of the DB data and access to which is required to enforce SI. The SI data is updated by triggers in transactions which update the DB. There are some problems with this approach. Changes in SI constraints require incorporation of changes in existing update transactions, modifying previously existing SI data and (possibly) creation of new SI data. This scheme requires the updating transactions to have knowledge of what SI data

exists in the system and which of them need to be updated. Since the updating transactions are user written they are unreliable and hence may not properly update the SI data, thus making the whole mechanism unreliable. Hence the author's claim that the separation of the SI data from DB data can be expected to increase the reliability of SI enforcement does not seem to be justified. Depending on the SI constraint, the volume of the SI data may be quite significant in relation to the DB data. Hence the author's claim that the size of SI data would be small would be true only for certain constraints.

The mechanisms proposed for specification and enforcement of integrity and security constraints would have a great impact on most other features/mechanisms/aspects of DDBS and DDB. Hence the design of these mechanisms cannot be done in isolation or without considering the design of other mechanisms in the DDBS. The realization of these fundamental facts is essential before one could hope to come up with viable and efficient solutions for handling these problems.

2.2 Analysis Of Deadlock Detection Algorithms

Quite a few proposals have been published in the literature to handle this problem (An excellent review of the deadlock problem in operating, data base and distributed systems can be found in [Marsland 1978]). Recently two apparently effective solutions have been proposed [Isloor 1978, Menasce 1978b]. Criticism of earlier proposals can be found in [Isloor 1978].

2.2.1 Distributed Protocols -

The authors of [Isoor 1978] have given the following as the advantages of their scheme over that of others':

1. Synchronization problems minimal or non-existent.
2. Reduced computation and communication overhead.
3. Avoids message congestion.
4. Processes allowed to have more than one outstanding request for resources. Thus more general and realistic than previous proposals.
5. Only "on-line" deadlock detection algorithm to be proposed until now.

I have uncovered a problem in this algorithm. The authors have stated: "In our 'on-line' detection of deadlocks, the installation which decides whether or not to grant a request for a data resource residing on its computer can, as a consequence of its decision, discover a deadlock without further delay". They have further stated that their scheme avoids the synchronization problems due to communication delays.

Unfortunately these statements are not correct. The algorithm fails to detect a deadlock in the following situation. Let us start with the situation as depicted in Figure 1a. Now let us assume that in C1 (Controller 1) P2 (Process 2) requests D1 (Data item 1). C1 updates its graph by adding the PRW (process request wait) edge (P2,D1), sends this information to C2 and checks its graph and finds no deadlock. C1's idea of the system graph at this moment is depicted in Figure 1b. When the above is happening at C1, at C2 P3 requests D4. C2 updates its graph by adding the PRW edge (P3,D4), sends this information to C1, and checks its graph and finds no deadlock. C2's idea of the graph at this moment is depicted in Figure 1c. After certain amount of time C2 receives C1's message and C1 receives C2's message. Now C1 and C2 update their graphs to get the one depicted in Figure 1d. Clearly this graph has a cycle. But after this updating C1 and C2 won't check for deadlock (this follows from the above quoted statement of the authors). Thus although a deadlock exists it does not get detected. This problem arises because of the communication delays in transmitting changes to the system graphs and because the changes are computed and transmitted asynchronously by the nodes.

Some approaches to overcoming this problem are:

Each controller could check for deadlock whenever it updates its graph (not only when the update is due to a decision made by that controller itself). The problem with this approach is that all the nodes would detect a deadlock condition (almost simultaneously) when it arises. Only one of these nodes should be allowed to initiate recovery actions. Achieving that is not likely to be easy.

Naturally the following question arises: Why not maintain the graph in only one node (call it the DD (deadlock detection) node)? Any time that node updates its graph (due to decisions made at other nodes or at the same node) it should check the graph for deadlock conditions. Since updates to the graph have to be sent to only the DD node, the traffic on the network would drop tremendously (compared to the scheme of [Isloor 1978] where updates to the graph are sent to all nodes). Of course there is a reliability problem with this approach. There is the possibility of the DD node crashing. To take care of this problem, one could think of having a primary DD node and

secondary DD nodes, with the secondaries being ordered linearly. The graph would be maintained at the primary and secondary DD nodes. But deadlock detection would be done only by the primary node. If the primary goes down, the first secondary in the linear order would become the primary. When the crashed DD nodes are back in service they have to copy the current graph from the current primary.

A centralized scheme has been proposed in [Gray 1978]. In Distributed INGRES also Gray's scheme is to be used [Stonebraker 1978]. In [Menasce 1978b] it has been pointed out that while the centralized method may be practical and efficient for local networks it may impose fairly large communication costs in geographically distributed systems. This observation would be correct only if in the geographically distributed system broadcasting is not used for message transmission. Merely distributing the DD function does not automatically mean greatly reduced communication costs. Many subtle problems may be introduced due to distribution. The scheme of [Isloor 1978] had distributed deadlock detection and also some flavour of a centralized scheme (since the whole graph was maintained - in fact in all nodes). I have already illustrated the problem with that scheme. The storage requirements (for the graph) are going to be much larger for the algorithms of [Isloor 1978] than those for the protocols of [Menasce 1978b], not only because only a part of the global graph is maintained at each node in the latter scheme but also because the latter scheme's graphs do not have data nodes in them. Now let me analyze the latter schemes and show some of the problems with them.

[Menasce 1978b] has presented two protocols - a hierarchical one and a distributed one - for DD. Gray's scheme is a degenerate case - a single level hierarchy with only one non-leaf controller - of this hierarchical protocol. In fact, Gray mentions, in passing, the possibility of developing the more general version of his scheme. In the hierarchical protocol each process is permitted to wait for more than one resource simultaneously. It is not definite if this is allowed in the distributed protocol. If it is allowed then a problem arises in using this protocol. It is possible for the same deadlock condition to be detected by more than one node. This is a highly undesirable feature of this protocol (I have already pointed out that one way of improving the schme of [Isloor 1978] would also lead to this type of a situation). The following sequence of events would illustrate my observation about this protocol. Let us consider a network of 4 nodes - S1, S2, S3 and S4 - and 3 transactions - T1, T2 and T3. Sorg(Ti) is the node (site) of origin of Ti.

Let Sorg(T1) = S1
Sorg(T2) = S2

Sorq(T3) = S3

Initially T3 is executing. T2 gets some resource at S2, executes for a while and then sends requests to S3 and S4. The resource required at S3 is being currently used by T3. The resorce request at S4 is granted. Now S3 creates the arc T2 -> T3 in its graph and sends the pair (T2,T3) to S2 (meaning that T2 is waiting for T3). Now the graphs are:

```
At S2: T2 -> T3
   S3: T2 -> T3
```

Now let us assume Tl starts at Sl and sends requests to S2 and S4. S2 finds that the resource required by Tl is being used by T2. So it adds the arc Tl -> T2 to its graph, and sends the pair (Tl,T3) to Sl and S3. So the graphs now become:

```
At Sl: Tl -> T3
   S2: Tl -> T2 -> T3
   S3: T2 -> T3
       Tl -> T3
```

While Tl's request to S4 is being processed it is found that the required resource is being used by T2. So S4 creates the arc Tl -> T2 and sends the pair (Tl,T2) to Sl and S2. Now the graphs are:

```
At Sl: Tl -> T3
       Tl -> T2
   S2: Tl -> T2 -> T3
   S3: T2 -> T3
       Tl -> T3
   S4: Tl -> T2
```

Now T3 sends a request to S4. S4 finds that the required resource has been locked by T2. So it creates the arc T3 -> T2 and sends the pair (T3,T2) to S3 and S2. When the pair is received at S2 and S3, both nodes discover a cycle in their graph and thus both detect the same deadlock situation. Whatever mechanism may be used to resolve deadlock situations, it is not at all desirable for a deadlock to be detected independently by more than one node in the network. Distribution of the DD mechanism should not result in these kinds of situations.

The above type of situation can be avoided if a transaction is allowed to wait for only one resource at a time. This would mean that a transaction has to send a lock request, get the response and only if the request is granted send the next lock request. This is too great a restriction and too great a price to pay for achieving distributed DD.

This would decrease response time of transactions greatly, if it is ever implemented. Decrease in response time would occur even for local transactions. This restriction would not reflect the way the distributed query algorithms published in the literature are meant to be implemented. Any practical and effective DD scheme should permit more than one lock request to be dispatched simultaneously. There is a greater need for this flexibility if updates to redundant data are to be handled efficiently.

2.2.2 Hierarchical Protocol -

After reading the above analysis and the referenced papers one may feel that probably the best approach to DD is the hierarchical protocol of [Menasce 1978b]. There are some problems with this protocol too. Firstly, according to the authors, this protocol is to be used only when there is no redundant storage of data (To me it is not clear as to why this restriction is being imposed - it appears that the protocol would work even otherwise). Secondly, if a deadlock is detected by the algorithm of [Isloor 1978], it is very easy to identify the processes responsible for the deadlock (since the reachability sets are maintained at all nodes). But in the hierarchical protocol direct identification of the processes is possible only when a deadlock is detected by a leaf controller (LK). When a deadlock is identified by a non-leaf controller (NLK), the LKs of the subtree with that NLK as the root must be examined (since the NLKs maintain only the IOP graph and not the full graph of the LKs) to determine the processes involved in the deadlock. This identification, it appears, would be more difficult for the distributed protocol. Potentially all nodes may have to be enquired to achieve that. The problems in the last two cases are caused because in these cases the cycles in the transaction wait for (TWF) graph would be a condensation of the global TWF graph. The authors have not considered the identification problem at all. This is an important problem and the authors of future proposals should not neglect it. Some additional message traffic would be generated to achieve this identification.

Thirdly, the reliability of the hierarchical protocol is very bad. The authors have not discussed it. One could get into serious trouble if the LKs or NLKs crash. Depending on the level of the crashed nodes in the hierarchy, DD, identification of involved processes and deadlock resolution may become impossible. The authors have stated that the design of the distributed protocol was motivated by the desire to support reliable operation in environments subject to failures. I find the reliability of the distributed protocol also to be not very good. While node failures may not seriously jeopardize DD, the identification and deadlock resolution capabilities may be seriously impaired. The

consequences of network partitioning would be worse (for both the protocols) than those of node failures. If we ignore for a moment the previously mentioned problem with the algorithms of [Isloor 1978], one would find that those algorithms provide the maximum reliability (since the complete graph is maintained at all nodes).

Neither have the authors of [Menasce 1978b] done a complete job in specifying the protocols. They have not stated as to how the TWF graphs and particularly the IOP graphs would be modified when a resource is released (as a result of the completion of a transaction or due to its preemption to break a deadlock). Atleast the changes to be made to the IOP graphs would be non-trivial. The algorithms of [Isloor 1978] take care of resource releases also. In the description of the distributed protocol, it has been stated that if there are multiple copies of data, lock requests have to be sent to all controllers which keep a copy of the data. I cannot understand as to why this requirement must be placed on even read lock requests.

2.2.3 Centralized Protocol -

As already pointed out the designers of Distributed INGRES propose to use a centralized DD scheme (the DD node is termed the SNOOP). Each site has a concurrency controller (CC) for local transactions. Only the SNOOP needs to be informed of "wait for" conditions. This means that each CC sends "wait for" conditions to the SNOOP. It has been stated that when a master completes a transaction, it sends a "done" to the SNOOP, who can appropriately update his "wait for" graph. This last one is not the right action to perform. The master should not send a "done". Each CC should send a "done" as soon as a resource is released. Otherwise problems would arise, since the graph would be updated pretty late - as a result some non-existing deadlocks may be detected by the SNOOP. Moreover the master merely sending a "done" to the SNOOP would not furnish the SNOOP with enough information to update the graph. Only the CCs can tell which of the currently waiting transactions has been granted the resource released by a terminating transaction. These arguments also apply to the "reset" message that a master is supposed to send the SNOOP if it finds that some sites have not responded. For these reasons, the ALGORITHM MASTER and ALGORITHM SLAVE given in [Stonebraker 1978] must be modified to reflect the required changes.

There is also a problem in the case of the failure of the current SNOOP. ALGORITHM RECONFIGURE states that a new SNOOP is calculated by a prearranged algorithm and that the local conflict graph is sent by each node to that site. If in addition to the SNOOP (which itself would most likely be a data node also) some other nodes have also crashed or if

there is a network partitioning then the new SNOOP would not be able to get a complete copy of the global "wait for" graph (since it would not be able to communicate with some nodes). The consequences of trying to detect deadlocks with a partial global graph would be bad. Hence I find the reliability of the DD mechanism of Distributed INGRES to be not good.

2.3 Performance And Correctness Evaluation Of Mechanisms

In the past not much attention has been given to the evaluation of the performance of the various mechanisms proposed for dealing with different DDBS problems. This has been taken note of, recently, by some researchers. [Bernstein 1978b] has stated that formal models for measuring the amount of concurrency allowed by and the amount of communications traffic generated by a synchronization mechanism would provide a sounder basis for comparing and selecting concurrency controls. [Edelberg 1978] has stated that further research is needed for the performance evaluation of proposed concurrency control and global data manipulation mechanisms via simulation, analysis or experimentation. Chandy, in his survey [Chandy 1977] of performance models in distributed systems, has characterized the modeling efforts in the DDB area as being in the first stage of development. This stage is characterized by models which focus on relatively narrow subproblems - those that do not generally consider the "big picture". He has noted that there seems to be a great need to combine correctness (logical) models and predictive (analysis) models, and to include these models in the process of synthesis (optimization).

In the following subsections I have made a critical evaluation of some of the proposed concurrency control and crash recovery mechanisms in terms of their performance, correctness, etc. Some of the analytical and simulation efforts, reported in the literature, have also been reviewed. The previous two subsections (2.1 and 2.2) also contain material relevant to this subsection's subject matter. The DDBS SDD-1's mechanisms have been evaluated in [Mohan 1979a].

2.3.1 Analytical And Simulation Results -

One of the significant contributions on this topic is the work reported in [Gelenbe 1978]. A formal model and certain performance measures (like promptness, coherence, etc.) have been designed to analyze the trade-offs involved in choosing update control protocols in fully redundant data base environments. That paper contains a good description of

different performance measures for a DDBS. Two families of update control techniques have been analyzed, with respect to the promptness and coherence measures. The first parameterized family of policies is based on postponing the application of each update for a fixed time (R) with the intention of waiting notice of any nearly simultaneous updates. The second family of policies avoids undoing updates entirely by incorporating an update only when it is known to be the oldest update not yet applied at the center. A parameter S, of the latter family, is the time between broadcasts of status messages (about the update activity during a certain period of time) by each center. The analysis gives the trends in the variation of promptness and coherence as S/R is varied (excepting for these, the rest of the results are ones that are very intuitive). It has been stated that S should be closer to the mean transit delay (the time between dispatch of an update message and its subsequent receival) than to the mean time between update originations at a center. If this policy were to be followed then the resulting message traffic would be enormous. An assumption, of the whole analysis, which is very difficult to justify is that the system reaches equilibrium. The authors have made no attempt to do that.

Since the analysis was done under very restrictive (unrealistic) assumptions it is not at all obvious as to how the results would change when the restrictions are relaxed to reflect a real life system's characteristics. Further, promptness and coherence are not the only tradeoffs to be considered in choosing an update control policy. Time to confirm succesful update application (i.e. response time) and traffic induced on the network by the mechanisms are also some of the other factors that need to be considered.

Similar in flavour to the above is the work on analytical and simulation evaluations of update control algorithms (for fully redundant data) reported in [Garcia-Molina 1978]. The algorithms that have been considered are the ones described in [Ellis 1977, Thomas 1978] and a centralized one. This work has been directed at comparing the performance of different algorithms and not at predicting the exact performance of a given system. This is a consequence of the extensive number of restrictive assumptions that have been made in building the simulator.

Response time has been used as the primary performance measure. It is defined to be the difference between the finish time and the time when the update arrived at the originating node (where finish time is the time when the originating node has finished all work on the update; notice that at that time other nodes might still not be done with that update). The results indicate that the centralized algorithm performs considerably better than the distributed voting algorithm [Thomas 1978], in most cases. Further, the

Ellis ring algorithm [Ellis 1977] and its variants are shown to perform worse than the distributed voting algorithm under most circumstances. These results must be interpreted with great caution since the reliability aspects of the algorithms have not been considered. The results are valid only when there are no link/node failures or message losses.

Garcia-Molina has also analyzed the stability characteristics of what are termed deferring and rejection algorithms. In a deferring algorithm, when a transaction is attempting to obtain its locks and finds an item that is locked, the transaction is deferred until the item becomes available. When the item becomes available, the transaction obtains the lock and continues the locking process. In the case of the rejecting algorithm, transactions are rejected whenever they encounter a locked item. When the transaction is rejected, it releases all the locks it holds, waits for some time and then restarts the locking process from scratch again. The centralized and Ellis algorithms are examples of the deferring algorithm, while the distributed voting algorithm could be considered to be an exampple of the rejecting algorithm. It has been shown that with finite number of users the deferring algorithm would always be stable, whereas the rejecting algorithm might be unstable (i.e. the response time may become infinite).

In general the aim of the above efforts has been to understand the functioning of some algorithms rather than to carry out a parameter study, with a view towards optimizing certain aspects of their operation. A comparison of the update algorithms of [Thomas 1978] and [Ellis 1977] can be found in [Pardo 1977] also.

Message Count as Performance Measure

In the DDBS literature there has been a tendency to evaluate concurrency control mechanisms by comparing the number of messages generated in executing a transaction using the various mechanisms under consideration (see [Badal 1978b, Rothnie 1977, Stonebraker 1978]). While the number of messages do have a great impact on the message traffic in the network, the lengths of the messages are also of more or less equal importance (this importance depends on the message transmission characteristics - whether it is packet or message switched - of the underlying network). This fact must be borne in mind while comparing different schemes. Scheme A requiring lesser number of messages than Scheme B does not necessarily imply that the time spent in message transmission in the former is definitely smaller than that in the latter. In view of this observation I cannot accept, without question, the assertion made in [Badal 1978b] that their distributed concurrency control (for partially redundant data bases) protocols' overall performance is considerably improved over other proposed protocols.

The authors of [Badal 1978b], in their attempts to reduce the number of messages to be sent to achieve synchronization, seem to have introduced the possibility of lot of information being sent more than once to a particular site. This happens in the case of SUM messages. Under normal conditions these duplicate message transmissions cause a wastage of network transmission capacities. Also, they make the message sizes unpredictable. Something that is very crucial to the efficient working of these protocols, in real-life, is being able to determine the proper value of Tmax (Tmax is the maximum average delay between a sender and several destinations). This would be very difficult for the following reasons: There would be a very great variability in the sizes of messages (as already pointed out), particularly those of the REQ, REQ-ACK and READ COMMAND messages (since each of them would include variable amounts of different pieces of information like acknowledgements, write messages, list of rejected transactions, duplicate SUM messages, etc.). Hence the time taken for sending each of those messages would vary tremendously. Thus the variance of the delay in sending a message could conceivably be quite high.

Tmax is not a function only of the distance between the different sites. A low estimate of Tmax could lead to a heavy amount of repeated sending of messages (particularly REQ messages), since the sending sites would time out fast - even before the intended recipient gets a chance to acknowledge. This leads to inefficient usage of the communication network and the processing capacities at the different sites, and to lower transaction throughput. A high estimate of Tmax could lead to delayed detection of failures, and the consequent delay in the already initiated transactions' execution completion or rejection, initiation of recovery mechanisms and processing of new transactions. This could again lead to lower throughput. Finding an optimal value for Tmax, even through simulation, would be a difficult proposition because of the difficulties involved in being able to characterize the lengths of the messages mentioned above.

The choice of preferred read sites should be made judiciously. If only a small fraction of the existing sites are chosen as preferred read sites, then these sites would become a bottleneck, since enormous number of messages (mainly REQ-ACKs) would be sent to these few sites and they would have to process them fast. This would result in decreased throughput. Somehow all the sites should be made to receive more or less equal number of messages.

2.3.2 Correctness And Other Considerations -

Read-Driven Protocols

One of the positive aspects of the proposal of [Badal 1978b] is that it does not require the storage of time stamps along with data items (the latter requirement imposes a heavy overhead in most of the other protocols proposed in the literature). The described protocols are 'read driven', in the sense that the transmission of updates is demanded by subsequent reads (There is no notion of explicit locking of data resources). The consequence of this is that the processing of read requests gets delayed considerably, in order to complete the incorporation of the previous updates. In an on-line enquiry and update environment these delays to answer queries may be intolerable. There may be situations where such a degraded response time is unacceptable.

One of the properties of the above protocols is that, if there are no crashes (link/site failures) then no transaction would ever be rejected. On the contrary, SDD-1's protocols may have to reject transactions (READ messages), even under those perfect conditions. In the event of link/site failures the former protocols would permit non-serializable execution of transactions. This may be a very undesirable feature of these protocols. In deciding to reject a transaction the protocols seem to be doing some unnecessary work. If a preferred read site of a transaction is found to be non-responding I don't see any reason as to why the other read sites should try to select a new preferred read site, instead of them informing the initiating site and (probably) each other that that transaction is being rejected. Since one of the read sites of the transaction (namely the original preferred read site) is not responding, there is no way that the transaction could be executed. So there is no point in selecting a new preferred read site.

While discussing recovery actions that need to be taken on partition reconnection, the authors have not stated as to how exactly the local logs (generated during the time the network was partitioned) would be compared, what comparison really means, how to determine what transactions would have to be rerun and what would be the criterion for deciding how far the data base state should be rolled back in order to restore consistency. The protocols to be followed to achieve global roll back have not been described. Until these are clearly spelt out and are found to be correct one cannot support the claim that the proposed protocols provide automatic recovery to a consistent state in case of network partitioning. Being able to restore a consistent state is critically dependent on the correctness of these, as of now unspelt, procedures.

Further, the authors have stated that a transaction can determine whether any of its Read or Write events would be executed using potentially inconsistent data due to missing Read or Write events from non-responding sites. My contention is that a transaction cannot determine that fact. A transaction would have to assume that potentially all its Read and Write sites are inconsistent. This is because a non-responding site's write messages might have been sent to some nodes but might not have been sent to the other nodes to which they were supposed to have been sent. The transaction in question cannot determine what the intentions (in terms of write sites) of the parially executed transaction in the non-responding site were. It is quite possible that the partially executed transaction had intended to send write messages to all the other sites in the network.

Another matter that has not been discussed is the generation of time stamps for transactions (Nothing similar to the Time Stamp Generation Rule of [Thomas 1978] has been given). It has been implicitly assumed (without being stated in the paper) that once a site (say Si) responds to a REQ message (whose time stamp is TSj) from another site (say Sj) with a REQ-ACK, then any transaction initiated in the future by Si would be given a TS value greater than TSj (this means that the clock at every site would be advanced to the TS of an acknowledged REQ, if the latter is greater than the current clock time at that site). It is very important to recognize this assumption because, when network partitioning occurs this clock synchronization would be highly jeopardized and that would have a great impact on the ability to restore consistency after partition reconnection (A discussion regarding this point with respect to SDD-1, can be found in [Mohan 1979a]).

Distributed INGRES Protocols

The concurrency control mechanisms for Distributed INGRES have been presented in [Stonebraker 1978]. A set of performance algorithms (which have some data consistency problems) and a set of so called reliable algorithms (which have low performance) have been proposed. No time stamps are used by these mechanisms. In comparing these mechanisms with some others, Stonebraker has used the number of messages generated for each transaction and estimates of response time as the measures of comparison. While the Distributed INGRES mechanisms may appear to be superior to the others with respect to these measures, what is not obvious is the amount of concurrency permitted by the former. Further, there is no discussion of how the local concurrency controllers (at the different sites) perform their functions. Unless information about that is available, one cannot be sure (even in the absence of crashes) that the interleaved concurrent execution of different transactions (accessing or modifying distributed data) would be

serializable. In addition, it has not been demonstrated that the reliable algorithms do not avoid the third data inconsistency problem that the performance algorithms have. The emphasis seems to have been to guarantee the incorporation of updates at all sites or its rejection by all sites, by using a two-phase protocol. Synchronization to achieve serializability does not seem to have been given the needed importance (in contrast to, for e.g., in SDD-1).

Ticket-Based Protocols

[Le Lann 1978] has presented some algorithms for data sharing, which use the notion of 'tickets'. These algorithms are discussed in the context of what the author terms as integrated and partitioned architectures. Unfortunately the differences between the two types of architectures are not well explained. I have been informed (by Le Lann in a personal communication) that integrated is equivalent to fully-redundant and partitioned to either strictly partitioned or partially redundant. Le Lann has stated that in an integrated architecture deadlock avoidance or detection is to be performed locally by every controller and that conventional (centralized) techniques can be used for that purpose. I don't see as to how the conventional techniques would be sufficient in the case of the integrated architecture. Since, even in the latter architecture, locking would need to be done to update the copies, a distributed algorithm would be needed to implement deadlock detection or avoidance mechanisms.

Further, it has been stated that in a partitioned architecture if only one controller at a time is issuing allocation requests, there is no potential for deadlocks. This would permit only static claiming of data. The author has chosen to avoid deadlocks rather than detecting them, if and when they occur. This choice may not be an advisable one, considering the fact that in [Isloor 1978] it has been stated that to maintain the operational fidelity of any DDBS with respect to the problem of deadlock, detection techniques are may be more advantageous than prevention or avoidance methods (This view has been expressed in [Peebles 1978] also). As a result the achieved parallelism would be much less than the potential parallelism that would be existing.

While Le Lann has described how sequential identification of user requests is achieved (even in the presence of message losses), he has not explained well as to how the storage processors would make use of the identification in achieving internal consistency and mutual consistency, and in avoiding deadlocks. The discussion is very sketchy and is difficult to follow. Since the author has not compared his scheme with the other researchers' schemes, one encounters more difficulty in understanding this scheme.

[Gelenbe 1978] has listed some of the secondary goals that need to be considered in selecting an update control technique for a specific data base environment. They are: (1) reducing delays in applying updates, (2) avoiding favoring updates from some centers, (3) giving similar response to identical queries submitted simultaneously at different nodes, (4) minimizing the need to undo updates, and (5) insuring that the order in which updates are applied conforms with their actual order of submission.

With regard to deadlocks [Isloor 1978] has stated: "It is difficult to estimate the performance effects of deadlock detection or prevention in DDBS, since communication time is the critical factor. Once DDBS become a commercial reality experimental data can be gathered to measure the performance". See the previous subsection (2.2) for an evaluation of the performance and correctness of some recent deadlock detection schemes.

3.0 PARAMETERIZABLE ADAPTIVE FUTURE ARCHITECTURE

The pursuit for new and novel architectures needs to be encouraged. In this section I would like to present the salient features of a future DDBS architecture which is highly parameterized and adaptive to different environments. The motivations for this architecture proposal have also been presented.

3.1 Motivations And The Proposal

In the literature one finds many solutions to different DDBS problems. If one looks at the solutions/algorithms proposed for a particular system problem, it would be obvious that the degree to which the problem is solved by the solutions varies and there is a corresponding variation in the overhead/cost involved in adopting the solution also (see Section 2.3 for references on performance evaluation which illustrate this remark for the case of update synchronization algorithms). As a result, it is likely that one solution would be appropriate for a particular DDB environment, while another solution would be more appropriate for a different environment (see [Badal 1978b, Bernstein 1978a, Chu 1979, Garcia-Molina 1978, Gelenbe 1978, Hammer 1978, Hevner 1978, Isloor 1978, Le Lann 1978, Maryanski 1978b, Menasce 1978a 1978b, Pelagatti 1978, Shapiro 1978, Stonebraker 1978, Thomas 1978, Yeh 1978a] for the variety of environments assumed while the authors propose certain algorithms/solutions).

It may not be advisable at all, to use the most general solutions/algorithms in all environments. Invariably

generality would bring with it certain amount of inefficiency. Efficiency would be an overriding requirement in most DDB applocations. Hence I feel that the DDBS of the future should be designed and implemented so that they would be able to use many solutions/algorithms for solving a particular problem (like deadlock detection, concurrency control, query processing, resiliency/reliability, etc.). The data base designer or enterprise administrator using such a system should be able to select (by setting values for certain parameters while installing a DDB using the DDBS) algorithms that are most appropriate (in terms of their performance characteristics and the extent to which they solve the DDBS problems) for his particular environment/application. This means that I am proposing that the future DDBS should have a parameterizable architecture. This approach is much more desirable than the one in which the designers of the DDBS select (based, probably, on their own perception of the environment in which the DDBS would be used) and implement one particular algorithm for each DDBS problem, thereby forcing the users (i.e. the enterprise administrators) of such DDBS to live with those solutions without any choice or flexibility.

The conjucture is that architectures based on the above concept would be much more adaptive and hence very efficient for different types of environments. Even in a given environment, the adaptability of the architecture would help to satisfy the changes in requirements that may occur in that environment over a period of time. Systems built with this type of architecture would not only be able to adapt to diffrent environments but also to the characteristics/architecture of different communication networks. One would be able to take advantage of the peculiarities of the underlying network or the application environment and thereby optimize the performance of the DDBS. Figure 2 conveys in a simple minded manner the basic idea of my proposal. Since the architecture is adaptive, it would be highly portable.

3.2 Implications Of The Architecture

The result of this parameterization of the architecture is that what we get is essentially a family of distributed data base systems, each with slightly different characteristics. This type of an architecture can be realized by building the software in a highly modular fashion, with the software divided into components on the basis of functionality.

All possible combinations of the different parameters' values are not likely to be compatible. A particular value for one parameter may require that certain other parameter(s) should not have certain values. This could be

the result of our previous observation (in Section 2) that solutions for a certain problem affect the solutions for other problems. Hence a handbook may have to be prepared to tell the DBA as to what combinations of the parameter values are valid. The preparation of this handbook would require a thorough analysis of the properties of algorithms/solutions denoted by the different parameter values.

Some parameters could be meant for specifying:

1. Degree of reliability (e.g. number of spoolers used to guarantee delivery of a message in [Hammer 1978]).

2. Crash recovery (e.g. number of recent updates to be remembered in [Shapiro 1978]).

3. Level of network transparency (should distribution be visible or invisible at the query level?).

4. Speed of detection of failures.

5. Time to recover from failures.

6. Level of consistency to be maintained (see [Gray 1978] for discussion on the possible levels).

7. Should redundant storage of data be allowed?

8. Whether each query would access data in more than one node.

9. Whether deadlock avoidance or prevention or detection algorithms should be activated or not.

10. Whether the type of communication is point to point or broadcast.

11. Time between the receipt of an update message and its subsequent incorporation (see [Gelenbe 1978] for more details).

12. If the hierarchical protocol of [Menasce 1978b] is used for periodic deadlock detection, then the frequency at which information about input and output port connection is sent by the leaf controllers to non-leaf controllers.

4.0 CONCLUSIONS/FUTURE DIRECTIONS

To summarize, research needs to be done to accomplish the following:

1. Study relationships amongst algorithms and protocols proposed for query processing, update synchronization, resiliency mechanisms, integrity & security enforcement mechanisms, data distribution and directory management, and the communications network and application characteristics (similar to the analysis done in [Mohan 1979a]).

2. Evaluate the performance of the various algorithms in terms of their execution overhead, amount of inter-node communication, etc.

3. Design of integrity and authorization enforcement mechanisms.

4. Elucidate the facilities that should be provided by data dictionary systems.

5. Define responsibilities of Global Enterprise Administrator (GEA) and Local Enterprise Administrator (LEA).

6. Development of integrated tools for helping the GEA and LEAs.

7. Development of integrated methodologies for DDB and DDBS design.

8. Development of methodologies and tools for collection of statistics about data base usage, and network and host utilization, etc.

5.0 REFERENCES

1. Adiba, M., et. al. [1978] Issues in Distributed Data Base Management Systems: A Technical Overview, Proc. IV Int. Conf. on VLDB, September.

2. ANSI [1978] Distributed Systems "Reference Model" (Draft 4), ANSI/X3/SPARC Study Group - Distributed Systems, February.

3. Bachman, C. [1978a] Commentary on the CODASYL Systems Committee's Interim Report on Distributed Data Base Technology, Proc. NCC, June.

4. Bachman, C. [1978b] Domestic and International Standard Activities for Distributed Systems, Proc. COMPCON'78 Fall, September.

5. Badal, D. Z. [1978a] Data Base System Integrity, Proc. COMPCON'78 Spring, March.

6. Badal, D. Z., Popek, G. [1978b] A Proposal for Distributed Concurrency Control for Partially Redundant Distributed Data Base Systems, Proc. III Berkeley Workshop on Distributed Data Management and Computer Networks, August.

7. Bernstein, P., Shipman, D. [1978a] A Formal Model of Concurrency Control Mechanism for Database Systems, Proc. III Berkeley Workshop on Distributed Data Management and Computer Networks, August.

8. Bernstein, P. [1978b] A Note on Theoretical Problems in Distributed Database Management, IV Int. Conf. on VLDB Proceedings Supplement: Panelists' Statements, September.

9. Chandy, K. M. [1977] Models of Distributed Systems, Proc. III Int. Conf. on VLDB, October.

10. Chang, S.K., McCormick, B.H. [1977] Intelligent Coupling of the User to Distributed Data Bases, Tech. Rep. KSL-3, Univ. of Illinois at Chicago Circle.

11. Chu, W., Hurley, P. [1979] A Model for Optimal Query Processing for Distributed Data Bases, Proc. COMPCON'79 Spring, March.

12. CODASYL [1978] Distributed Data Base Technology: An Interim Report of the CODASYL Systems Committee, Proc. NCC, June.

13. Edelberg, M. [1978] Statement for VLDB Session 9: Distributed Data Bases, IV Int. Conf. on VLDB Proceedings Supplement: Panelists' Statements, September.

14. Ellis, C. [1977] A Robust Algorithm for Updating Duplicate Data Bases, Proc. II Berkeley Workshop on Distributed Data Management and Computer Networks, May.

15. Garcia-Molina, H. [1978] Performance Comparison of Update Algorithms for Distributed Databases, Technical Note 143, Stanford University, June.

16. Gelenbe, E., Sevcik, K. [1978] Analysis of Update Synchronization for Multiple Copy Data Bases, Proc.

III Berkeley Workshop on Distributed Data Management and Computer Networks, August.

17. Gray, J. [1978] Notes on Database Operating Systems, In Operating Systems - An Advanced Course, R. Bayer, et. al. (Eds.), Lecture Notes in Computer Science - Volume 60, Springer Verlag, New York (Also IBM Tech. Rep. RJ 2188).

18. Hammer, M., Shipman, D. [1978] An Overview of Reliability Mechanisms for a Distributed Data Base System, Proc. COMPCON'78 Spring, March.

19. Hevner, A. R., Yao, S. B. [1978] Query Processing in a Distributed System, Proc. III Berkeley Workshp on Distributed Data Management and Computer Networks, August.

20. INFOTECH [1977] Distributed Processing, INFOTECH State of the Art Report, Infotech International.

21. INFOTECH [1978] Future Networks, INFOTECH State of the Art Report, Infotech International.

22. Isloor, S., Marsland, T. [1978] An Effective "On-Line" Deadlock Detection Technique for Distributed Data Base Management Systems, Proc. COMPSAC'78, November.

23. ISO [1978] Provisional Model of Open-Systems Architecture, ACM SIGCOMM's Computer Communications Review, July.

24. Le Lann, G. [1977] Distributed Systems - Towards a Formal Approach, Proc. IFIP'77, August.

25. Le Lann, G. [1978] Algorithms for Distributed Data-Sharing Which Use Tickets, Proc. III Berkeley Workshop on Distributed Data Management and Computer Networks, August.

26. MacDonald, V. [1978] Domestic and International Standards Activities for Data Communications, Proc. COMPCON'78 Fall, September.

27. Marsland, T., Isloor, S. [1978] A Review of the Deadlock Problem in Operating, Database, and Distributed Systems, Tech. Rep. TR 78-5, Univ. of Alberta, Canada, December.

28. Maryanski, F.J. [1978a] A Survey of Developments in Distributed Data Base Management Systems, Computer, February.

29. Maryanski, F. J. [1978b] The Management of Redundant Data in a Distributed Data Base, Tech. Rep. TR CS 78-21, Kansas State University, September.

30. Menasce, D., et. al. [1978a] A Locking Protocol for Resource Coordination in Distributed Databases, Proc. ACM-SIGMOD Int. Conf. on Management of Data, May-June.

31. Menasce, D., Muntz, R. [1978b] Locking and Deadlock Detection in Distributed Databases, Proc. III Berkeley Workshop on Distributed Data Management and Computer Networks, August.

32. Mohan, C. [1978a] Survey of Recent Operating Systems Research, Designs and Implementations, ACM SIGOPS' Operating Systems Review, January (Also Tech. Rep. TR-75, Univ. of Texas at Austin).

33. Mohan, C. [1978b] An Overview of Recent Data Base Research, ACM SIGBDP's Data Base 10, 2, Fall (Also Tech. Rep. SDBEG-5, Univ. of Texas at Austin, April 1978).

34. Mohan, C. [1979a] An Analysis of the Design of SDD-1: A System for Distributed Data Bases, In Distributed Data Bases, INFOTECH State of the Art Report, Infotech International (Also Tech. Rep. SDBEG-11, Software and Data Base Engineering Group, Univ. of Texas at Austin, April).

35. Mohan, C., Yeh, R. T. [1979b] Distributed Data Base Systems - A Framework for Data Base Design, In Distributed Data Bases, INFOTECH State of the Art Report, Infotech International (Also Tech. Rep. SDBEG-10, Software and Data Base Engineering Group, University of Texas at Austin, April).

36. Mohan, C. [1979c] Some Notes on Multi-Level Data Base Design, Technical Report TR-128, Software and Data Base Engineering Group, Computer Sciences Department, Univ. of Texas at Austin, May.

37. Mohan, C. [1979d] Data Base Design in the Distributed Environment, Working Paper WP-7902, Software and Data Base Engineering Group, University of Texas at Austin, May.

38. Mohan, C. [1980] A Perspective of Distributed Computing - Models, Constructs, Methodologies & Applications, Working Paper DSG-8001, Distributed Systems Group, Department of Computer Sciences, Univ. of Texas at Austin, January.

39. Pardo, R., et. al. [1977] Distributed Services in Computer Networks: Designing the Distributed Loop Data Base System (DLDBS), Proc. Computer Networks Symposium, December.

40. Peebles, R., Manning, E. [1978] System Architecture for Distributed Data Management, Computer, January.

41. Pelagatti, G., Schreiber, F. A. [1978] Comparison of Different Access Strategies in a Distributed Database, Proc. Int. Conf. on Data Bases: Improving Usability and Responsiveness, August.

42. Rothnie, J., Goodman, N. [1977] A Survey of Research and Development in Distributed Database Management, Proc. III Int. Conf. on VLDB, October.

43. Rothnie, J. [1978] Remarks in Response to: "Issues in Distributed Data Base Management Systems: A Technical Overview", IV Int. Conf. on VLDB Proceedings Supplement: Panelists' Statements, September.

44. Schneider, H. [1978] Some Remarks on a Paper by Adiba, et. al. on "Issues in Distributed Data Base Management Systems: A Technical Overview", IV Int. Conf. on VLDB Proceedings Supplement: Panelists' Statements, September.

45. Severino, E. F., Hannan, J. [1977] Operational and Technological Issues in On-Line Data Bases, In On-Line Data Bases, Infotech State of the Art Report, Infotech International.

46. Shapiro, R., Millstein, R. [1978] Failure Recovery in a Distributed Data Base System, Proc. COMPCON'78 Spring, March.

47. Stonebraker, M., Neuhold, E. [1977] A Distributed Data Base Version of INGRES, Proc. II Berkeley Workshop on Distributed Data Management and Computer Networks, May.

48. Stonebraker, M. [1978] Concurrency Control and Consistency of Multiple Copies of Data in Distributed INGRES, Proc. III Berkeley Workshop on Distributed Data Management and Computer Networks, August.

49. Thomas, R. H. [1978] A Solution to the Concurrency Control Problem for Multiple Copy Data Bases, Proc. COMPCON'78 Spring, March.

50. Yeh, R., Chandy, K.M. [1978a] On the Design of Elementary Distributed Systems, Proc. III Berkeley

Workshop on Distributed Data Management and Computer Networks, August.

51. Yeh, R., Chang, P., Mohan, C. [1978b] A Multi-Level Approach to Data Base Design, Proc. COMPSAC'78, November.

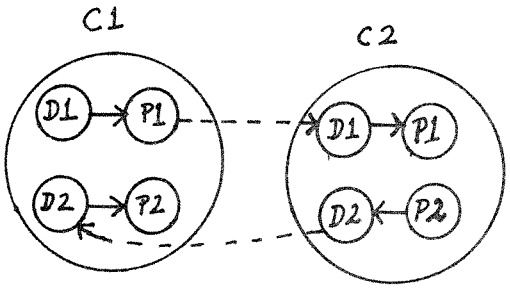


Fig. 1a

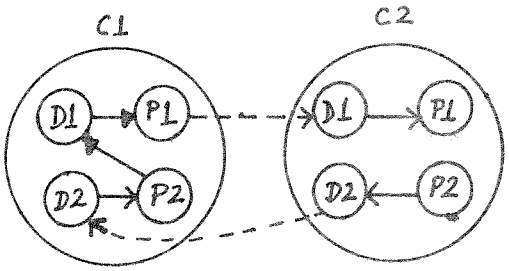


Fig. 1b

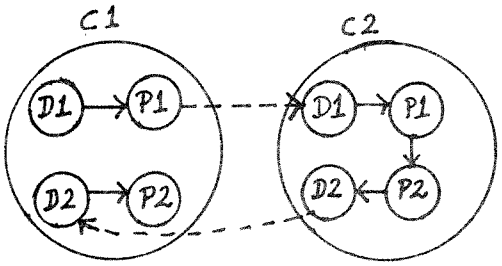


Fig. 1C

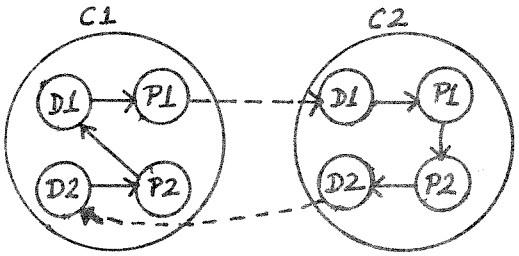
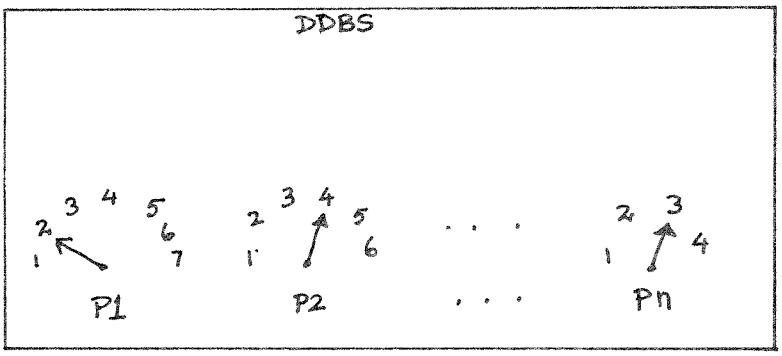


Fig. 1d



Pn denotes Parameter n

Fig. 2 DDBS with Parameterizable Adaptive Future Architecture "tuned" to a Particular Environment (Application and Network)