Old Dominion University Research Foundation

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

PERFORMANCE RELATED ISSUES IN DISTRIBUTED DATABASE SYSTEMS

By

Ravi Mukkamala, Principal Investigator

Final Report
For the period ended May 15, 1991

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NAG-1-1154
Mr. Wayne H. Bryant, Technical Monitor
ISD-Systems Architecture Branch

July 1991

Old Dominion University Research Foundation is a not-for-profit corporation closely affiliated with Old Dominion University and serves as the University's fiscal and administrative agent for sponsored programs.

Any questions or comments concerning the material contained in this report should be addressed to:

Executive Director
Old Dominion University Research Foundation
P. O. Box 6369
Norfolk, Virginia 23508-0369

Telephone: (804) 683-4293
Fax Number: (804) 683-5290

DEPARTMENT OF COMPUTER SCIENCE
COLLEGE OF SCIENCES
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA 23529

PERFORMANCE RELATED ISSUES IN DISTRIBUTED DATABASE SYSTEMS

By

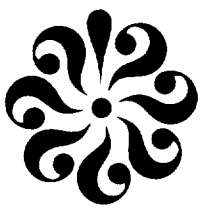
Ravi Mukkamala, Principal Investigator

Final Report
For the period ended May 15, 1991

Prepared for
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia 23665

Under
Research Grant NAG-1-1154
Mr. Wayne H. Bryant, Technical Monitor
ISD-Systems Architecture Branch

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508-0369



July 1991

Performance Related Issues in Distributed Database Systems

Ravi Mukkamala
Ajay Gupta

Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.

Final Report

Abstract

The key elements of research performed during the year long effort of this project are:

- Investigate the effects of heterogeneity in distributed real-time systems.
- Study the requirements of TRAC towards building a heterogeneous database system.
- Study the effects of performance modeling on distributed database performance.
- Experiment with an ORACLE based heterogeneous system.

1 Introduction

The main motivation for this project has been to investigate performance evaluation aspects of distributed systems with special focus on heterogeneous systems. To this end, we have conducted several studies to look at the practical and theoretical aspects of heterogeneous systems.

As part of theoretical studies, we looked at the effects of heterogeneity on scheduling in distributed systems. While simple randomized algorithms are claimed to effective in homogeneous systems, we found them to be ineffective in heterogeneous systems. To this end, we developed some new scheduling algorithms and evaluated their performance through simulation.

As part of practical studies, we have evaluated the database requirements of TRAC (TRADOC Analysis Command, U.S. Army). TRAC coordinates activities of projects which are geographically distributed throughout the country. This appeared to be an appropriate candidate for studies in heterogeneous distributed databases. To this end, we have studied this system and prepared a detailed proposal to that agency. This experience has highlighted several practical aspects of heterogeneous systems.

In addition, we have investigated low-cost schemes to build heterogeneous distributed systems. To this end, we experimented with PC-based Oracle databases. The results are interesting, but need further investigations before any strong conclusions may be drawn about the utility of the schemes.

We have also investigated some performance aspects of general distributed systems. More specifically, we have conducted a study to determine the effects of static locking in replicated distributed database system. Even though locking is less likely to occur in distributed systems, its effect on performance is found to be significant. We have extended earlier work on read-only transactions to update transactions.

Further, we have investigated the effect of distributed database modeling on transaction rollbacks in distributed systems. Whils uniform models seem to underplay the role of commutative transactions in improving the performance, more realistic models indicate a prominent role for these transactions.

In this report, we summarize our progress in these areas.

2 Scheduling in Heterogeneous Distributed Systems

In a distributed real-time system, jobs with deadlines are received at each of the nodes in the system. Each node is generally capable of executing a job completely. These jobs are scheduled for execution at the nodes by a scheduler. A distributed scheduler is a distributed algorithm with cooperating agents at each node. Basically, the agents cooperate through exchange of local load information. The decision for scheduling a job, however, is taken by a local agent. Much of the current work in distributed scheduling assumes identical scheduling algorithms, homogeneous processing capabilities, and identical request arrival patterns at all nodes across the distributed system.

As part of this work, we proposed a distributed algorithm to help schedule jobs with time constraints over a network of heterogeneous nodes, each of which could have its own processing speed and job-scheduling policy. We conducted several parametric studies. From the results obtained we conclude that:

- the algorithm has a large improvement over the random selection in terms of the percentage of discarded jobs;
- the performance of the algorithm tends to be invariant with respect to node-speed heterogeneity;
- the algorithm efficiently utilizes the available information; this is evident from the sensitivity of our algorithm to the periodicity of update information.
- the performance of the algorithm is robust to variations in in the cluster size.

In summary, our algorithm is robust to several heterogeneities commonly observed in distributed systems. Our other results (not presented here) also indicate the robustness of this algorithm to heterogeneities in scheduling algorithms at local nodes. In future, we propose to extend this work to investigate the sensitivity of our algorithm to other heterogeneities in distributed systems. We also propose to test its viability in a non-real time system where response time or throughput is the performance measure.

The results from this work are to appear in Lecture Notes in Computer Science, published by Springer-Verlag, 1991. Some of the results are to be presented at the 1991 Summer Simulation Conference at Baltimore, MD.

3 An Integrated Decision Support System for TRAC: A Case Study

In order to understand the requirements of real heterogeneous database needs of organizations, we studied the requirements of TRAC, an agency of the U.S. Army located in Fort Monroe, VA. Even though the project decision and management control office of TRADOC is centrally located at Fort Monroe, it interacts several agencies all over the country. Since each agency maintains its own database, it becomes impossible to integrate all the information. A heterogeneous database system seemed to be most appropriate in this environment. For this reason, we studied their current system, and proposed a cost-effective solution to build an integrated database system. A summary of the proposal is enclosed in the report.

4 Experiments with Oracle and DEC Databases

As part of this study, we investigated the feasibility of developing cost-effective schemes for heterogeneous database systems. In this context we experimented with Oracle databases. We developed software which can record all updates to a database externally in an ASCII file. Since on-line updation of remote files is expensive, in applications that only need periodic updates, we can easily send the ASCII files periodically to all relevant sites from the original updating site. When remote sites receive the ASCII file, they can execute our software which enforces the update at a remote site. The problem becomes complex when we consider heterogeneous database systems. But the proposed software solution with a standard format to record the updates can simplify the problems with the aid of dictionary mechanisms. In the report, we have enclosed the listings of this program.

We have also looked at a system employing two physically separate databases, but maintaining similar information. This system currently exists at the Navy information center in Norfolk. We propose a simple solution for maintaining consistency among the databases.

5 Reports and Publications

The results from this projected have so far resulted in the following publications:

1 R. Mukkamala, "Effects of Distributed Database Modeling on Evaluation of Transaction Rollbacks," *Proc. WSC'91*, December 1990, pp. 839-845.

2 O. Zein ElDine, M. El-Toweissy, and R. Mukkamala, "A Distributed scheduling algorithm for heterogeneous real-time systems," *To appear in Lecture Notes in Computer Science, Springer-Verlag Publications*, 1991.

3 M. El-Toweissy, O. Zein ElDine, and R. Mukkamala, "Measuring the Effects Heterogeneity on Distributed Systems," *To appear in Proc. of SSC-1991*, July, 1991, Baltimore, Maryland.

4 Y. Kuang and R. Mukkamala, "Performance analysis of static locking in replicated distributed database systems," *Proc. Southeastcon'91*, Williamsburg, VA, pp. 696-701.

N91-25953

MEASURING THE EFFECTS OF HETEROGENEITY ON DISTRIBUTED SYSTEMS

Mohamed El-Toweissy Osman ZeinElDine Ravi Mukkamala

Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529

ABSTRACT

Distributed computer systems in daily use are becoming more and more heterogeneous. Currently, much of the design and analysis studies of such systems assume homogeneity. This assumption of homogeneity has been mainly driven by the resulting simplicity in modeling and analysis. In this paper, we present a simulation study to investigate the effects of heterogeneity on scheduling algorithms for hard real-time distributed systems. In contrast to pervious results which indicate that random scheduling may be as good as a more complex scheduler, our algorithm is shown to be consistently better than a random scheduler. This conclusion is more prevalent at high workloads as well as at high levels of heterogeneity.

INTRODUCTION

With the advancing communication technologies and the need for integration of global systems, heterogeneity is becoming a reality in distributed computer systems. However, most existing performance studies of such systems still assume homogeneity; be it in hardware (e.g., node speed) or in software (e.g., scheduling algorithms). Generally, such homogeneity assumptions are dictated by the resulting simplicity in modeling and analysis.

Clearly, heterogeneous systems are less analytically tractable than their homogeneous counterparts. Typically, heterogeneity will result in increased number of variables in the context of analytical techniques such as mathematical programming, probabilistic analysis, and queuing theory. This is one reason for assuming homogeneity while using analytical techniques. In the case of simulation techniques, however, it is possible for a modeler to introduce any level of heterogeneity into the system. The problem now lies in the complexity of interpretation of the results. If the simulator was written with the basic objective of testing a hypothesis or comparing the performance of a set of algorithms, introducing heterogeneity will substantially increase efforts to separate its effects from those of the algorithm. Thus, a modeler is more likely to assume a homogeneous system.

With this in mind, we have been investigating into the effects of heterogeneity on the performance of distributed systems. Our initial efforts, reported in (ZeinElDine et al. 1991) and this paper, focus on scheduling in hard real-time systems. For this purpose, we have designed a distributed scheduler aimed at handling various heterogeneities; in particular, heterogeneities in nodes, node traffic and local scheduling algorithms.

In the rest of this paper we present the system model. Next, we discuss some issues related to the effectiveness of our algorithm. Major results with their respective conclusions are then portrayed. Finally, we highlight some recommendations for future work.

THE PROPOSED MODEL

For the purposes of scheduling, the distributed system is modeled as a tree of nodes as is shown in Figure 1. The nodes at the lowest level (level 0) are the *processing* nodes while the nodes at the higher levels represent *servers* (or guardians). A processing node is responsible for executing arriving jobs when they meet some specified criteria (e.g., deadline). The processing nodes are grouped into *clusters*, and each cluster is assigned a unique server. When a server receives a job, it tries to either redirect that job to a processing node within its cluster or to its guardian. It is to be noted that this hierarchical structure could be logical (i.e., some of the processing nodes may themselves assume the role of the servers).

The system model has four components: jobs, processing nodes, servers, and the communication subsystem. A job is characterized by its arrival time, execution time, deadline, and priority (if any). The specifications of a processing node include its speed factor, scheduling policy, external arrival rate (of jobs), and job mix (due to heterogeneity). A server is modeled by its speed and its node assignment policy. Finally, the communication subsystem is represented by the speeds of transmission and distances between different nodes (processing and servers) in the system.

Operation

The flow diagram of the scheduling algorithm is shown in Figure 2. When a job with deadline arrives (either from an external user or from a server) at a processing node, the local scheduling algorithm at the node decides whether or not to execute this job locally. This decision is based on pending jobs in the local queue (which are already guaranteed to be executed within their deadlines), the requirements of the new job, and the scheduling policies (e.g., FCFS, SJF, SDF, SSF etc. (Zhao et al. 1987)). In case the local scheduler cannot execute the new job, it either sends the job to its server (if there is a possibility of completion), or discard the job (if there is no such chance of completion).

The level-1 server maintains a copy of the latest information provided by each of its child nodes including the load at the node and its scheduling policy. Using this information, the server should be able to decide which processing nodes are eligible for executing a job and meet its deadline. When more than one candidate node is available, a random selection is carried out among these nodes. If a server cannot find a candidate node for executing the job, it forwards the job to the level-2 server.

The information at the level-2 server consists of an abstraction of the information available at each of the level-1 servers. This server redirects an arriving job to one of the level-1 servers. The choice of candidate servers is dependent on the ability of these servers to redirect a job to one of the processing nodes in their cluster to meet the deadline of the job. (For more details on operation and information contents at each level, the reader is advised to refer to (ZeinElDine et al. 1991)).

EXPERIMENT

In order to utilize the proposed scheduler as a vehicle for our research on measuring the effects of heterogeneity, first it has to be proven effective. Consequently, we have conducted several parametric studies to determine the sensitivity of our algorithm to various parameters: the cluster size, the frequency of propagation of load statistics (between levels), the processing node scheduling policy (FCFS, SJF etc), the communication delay (between nodes), and the effects of information structures. For lack of space, we present a sample of the results pertaining to the first three of these parameters. Accordingly, all the results reported here assume:

- the total number of processing nodes is 100;
- equal load at all nodes;
- communication delay between any nodes is the same.

The performance of the scheduler is measured in terms of the percentage of jobs discarded by the algorithm (at levels 0, 1 & 2). The rate of arrivals of jobs and their processing requirements are combinedly represented through a load factor. This load factor refers to the load on the overall system. Our load consists of jobs from three types of execution time constraints (10, 50 & 100) with slack (25, 35 & 300) respectively.

Discussion

We now discuss our observations regarding the characteristics of the distributed scheduling algorithm (DSA) in terms of the three selected parameters. In order to isolate the effect of one factor from others, the choice of parameters is made judiciously. For example, in studying the effects of cluster size (Figure 3), the updation period is chosen to be a medium value of 200 (stat=200). For each parametric study we have two sets of runs, they differ in the local scheduling policy at the processing nodes; one set uses FCFS while the the other uses SJF.

Cluster size Cluster size indicates the number of processing nodes being assigned to a level-1 server. In our study, we have considered three cluster sizes: 100, 50, and 10. A cluster of 100 nodes indicates a centralized server structure where all the processing nodes are under one level-1 server. In this case, level-2 server is absent. Similarly, in the case of cluster of 50 nodes, there are two level-1 servers, and one level-2 server. For 10-node cluster, we have 10 level-1 servers. In addition, we consider a completely decentralized case represented by the *random* policy. In this case, each processing node acts as its own server and randomly selects a destination node to execute a job which it cannot locally guarantee.

The results are plotted in Figure 3. These results show that our algorithm is robust to variations in

cluster size. In addition, its performance is significantly superior to a random policy.

Frequency of updations The currency of information at a node about the rest of the system plays a major role in performance. Hence, if the state of processing nodes varies rapidly, then the frequency of status information exchange between the levels should also be high. In order to determine the sensitivity of the proposed algorithm to the period of updating statistics at the servers, we experimented with four time periods: 25, 100, 200 and 500 units. The results are summarized in Figure 4. From these results, the following observations are drawn:

- our algorithm is extremely sensitive to changes in period of information exchanges between servers and processing nodes;
- even in the worst case of 500 units, the performance of our algorithm is significantly better than the random policy.

Local scheduling policy Our third parametric study is concerned with the effect of the scheduling policy at the processing nodes. In this paper, we report the results of the runs conducted with all the processing nodes having the same scheduling policy; either FCFS or SJF. Later in our work, we plan to experiment with different mixes of local scheduling policies. We would also give each node the freedom to select its scheduling policy in order to determine the impact of node autonomy on the overall performance of the system. This issue is of crucial importance, since there is no scheduling policy that best fits all working environments.

Revisiting the results of Figures 3 and 4, we can observe the following:

- both FCFS and SJF behave similarly at light to moderate loads, while SJF is consistently better than FCFS at high loads;
- the percentage of discarded jobs sharply increases with the increase in the load factor for both the Random and FCFS policies. However, for SJF the rate of increase in the percentage of discarded jobs dramatically drops at high loads.

The reason for the above result is that, at light to moderate loads there is no build up at the processing node queues, consequently, the dominant factor is the jobs being processed at the node processors. This behavior is the same for all policies. However, when the queues start to build up, the respective queue policy prevails. Hence, for the SJF, the short jobs with their relatively small slack, will have a better probability of being executed.

EFFECTS OF HETEROGENEITY

So far, our concentration has been on gaining better insight into the behavior of our algorithm in order to assess its viability and suitability to be able to conduct further research. Proven effective, we return back to the objective for which the algorithm has been developed. The main goal of the current phase of our studies is measuring the effects of heterogeneity on scheduling in hard real-time distributed systems. For this purpose, we are pursuing multiple experiments to measure the effects of node heterogeneity (simply represented by node speed), heterogeneity in scheduling algorithms, heterogeneity in loads as well as other system heterogeneities. In this paper, we present the effects of node heterogeneity. (Results on other types will be reported in a sequel of papers).

We consider four different node speed distributions (het1, het2, het3, and hom). The homogeneous case (denoted by hom) represents a system with 100 nodes having the same unit speeds. The three heterogeneous case are represented by het1, het2, and het3. Each of these set are described by a set of <# of nodes, speed factors> pairs. The average speed factor for all distribution is 1.0, so the average system speed is the same. The three heterogeneous case differ in their speed factor variance, thus varying the degree of heterogeneity. While het3 represents a severe case of heterogeneity, het1 is more biased towards homogeneity.

The results are included in Figure 5. From these results, we observe that:-

- with our algorithm, even though the increase in degree of heterogeneity resulted in an increase of discarded jobs, the increase is not so significant. Hence, our algorithm appears to be robust to node heterogeneities.
- the performance of the random policy is extremely sensitive to the node heterogeneity. As the heterogeneity is increased, the number of discarded jobs is also significantly increased.

With the increase in node heterogeneity, the number of nodes with slow speed also increase. Thus, using a random policy, if a slow speed node is selected randomly, then the job is more likely to be discarded. In our algorithm , since the server is aware of the heterogeneities, it can suitably avoid a low speed node when necessary. Even in this case, there is a tendency for high-speed nodes to be overloaded and low speed nodes to be under loaded. Hence, the difference in performance.

CONCLUSION

In this paper, we have presented a distributed scheduling algorithm that can tolerate different types of system heterogeneity. Following, we have conducted several parametric studies with the objective of evaluating the effectiveness of our algorithm. Rendering its effectiveness, we have started pursuing our studies toward our goal of determining the impact of heterogeneity on the overall system performance. Our initial step has been reported here, and it concentrates on the effect of node heterogeneity. Some interesting results have been obtained. From these results, we reach the following conclusions.

- *Concerning the algorithm behavior:* the algorithm is robust to variations in the cluster size; besides, it efficiently utilizes the available state information; moreover, it is sensitive to the local scheduling policy at the processing nodes.
- *Concerning the effect of heterogeneity:* the performance of the algorithm tends to be invariant with respect to node heterogeneity; in addition, the algorithm has a large improvement over the random selection in terms of the percentage of discarded jobs.

Currently, we are studying the effects of heterogeneity in local scheduling algorithms and heterogeneities in loads on the performance of the overall system. With heterogeneities in scheduling policies, each node may autonomously decide its own scheduling policy (FCFS, SJF, etc.). Similarly, by load heterogeneities we let the external load at a node be independent of the other nodes. Similarly, each node may autonomously decide its resources and their speeds. We propose to measure the effects of such heterogeneities in terms of the response time and throughput. We conjecture that the performance of random policies will continue to deteriorate under these heterogeneities as compared to *even simple* resource allocation or execution policies.

ACKNOWLEDGEMENT

This research was sponsored in part by the NASA Langley Research Center under contracts NAG-1-1114 and NAG-1-1154.

References

[] Biyabani, S.R.; J.A. Stankovic; and K. Ramamritham. 1988. "The integration of deadline and criticalness in hard real-time scheduling." *Proc. Real-time Systems Symposium*, (DEC.), 152-160.

[] Chuang, J.Y. and J.W.S. Liu. 1988. "Algorithms for scheduling periodic jobs to minimize average error." *Proc. Real-time Systems Symposium*, (DEC.), 142-151.

[] Craig, D.W. and C.M. Woodside. 1990. "The rejection rate for tasks with random arrivals, deadlines, and preemptive scheduling." *IEEE Trans. Software Engineering SE-16*, no. 10(OCT.), 1198-1208.

[] Eager D.L.; E.D. Lazowska; and J. Zahorjan. 1986. "Adaptive load sharing in homogeneous distributed systems." *IEEE Trans. Software Engineering*, SE-12(May), no. 5, 662-675.

[] Rajkumar R.; L. Sha; and J.P. Lehoczky. 1988. "Real-time synchronization protocols for multiprocessors," *Proc. Real-time Systems Symposium*. (DEC.), 259-269.

[] Shin, K.G.; C.M. Krishna; and Y.H. Lee. "Optimal resource control in periodic real-time environments." *Proc. Real-time Systems Symposium*. (DEC.), 33-41.

[] Stankovic J. and K. Ramamritham. 1986. "Evaluation of a bidding algorithm for hard real-time distributed systems." *IEEE Trans. Computers*C-34, no. 12(Dec.), 1130-1143.

[] ZeinElDine, O.; M. El-Toweissy; and R. Mukkamala. 1991. " A Distributed Scheduling Algorithm for Heterogeneous Real-time Systems." To appear in *Lecturer Notes in Computer Science*, Springer-Verlag.

[] Zhao, W.; K. Ramamritham; and J. Stankovic. 1987. "Scheduling tasks with resource requirements in hard-real time systems." *IEEE Trans. Software Engineering* SE-13, no. 5(May): 564-577.

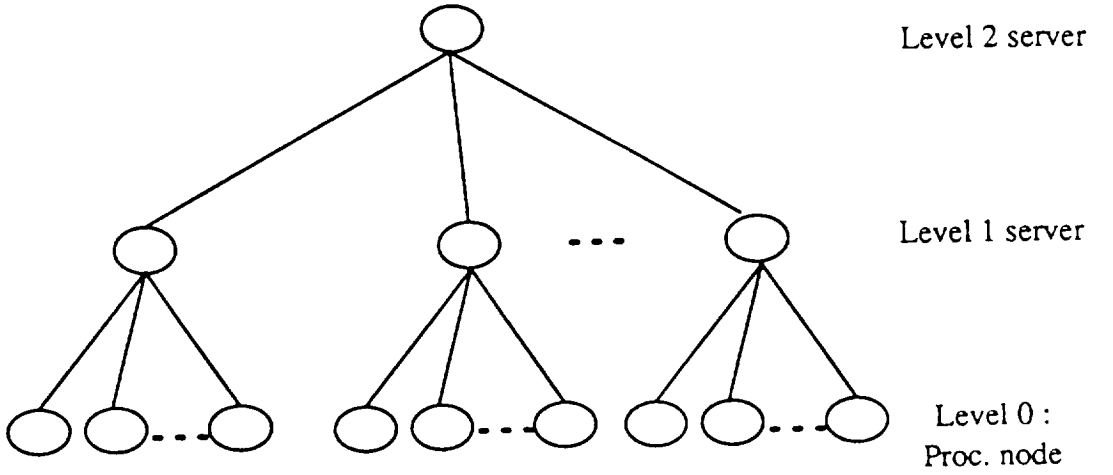


Figure 1 : System Model

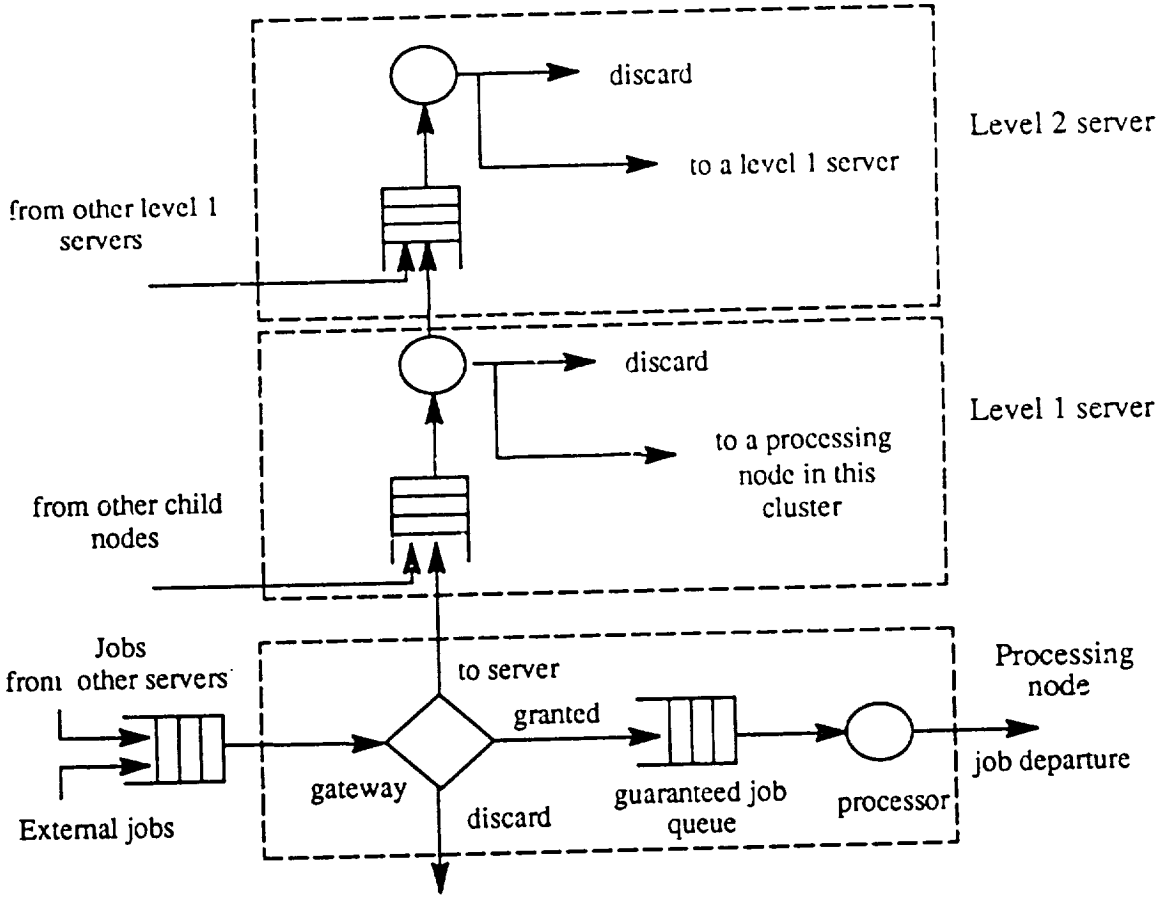
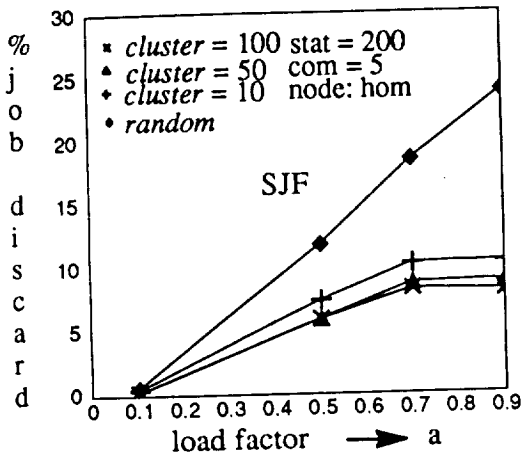


Figure 2 : Flow diagram of The Algorithm



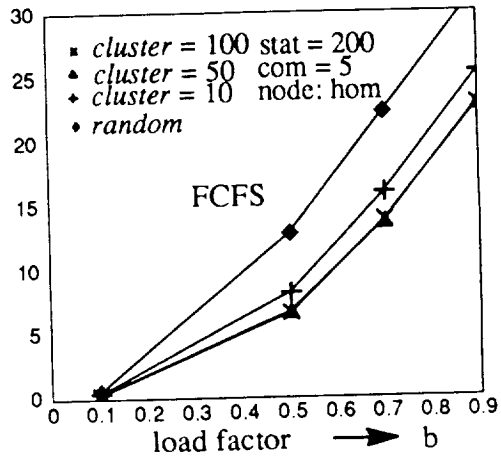
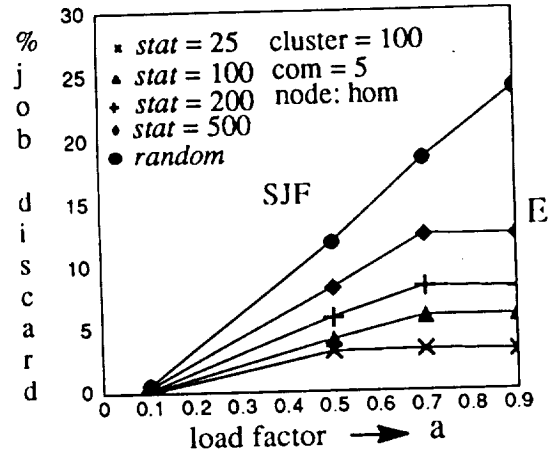


Figure 3:
Effect of cluster size



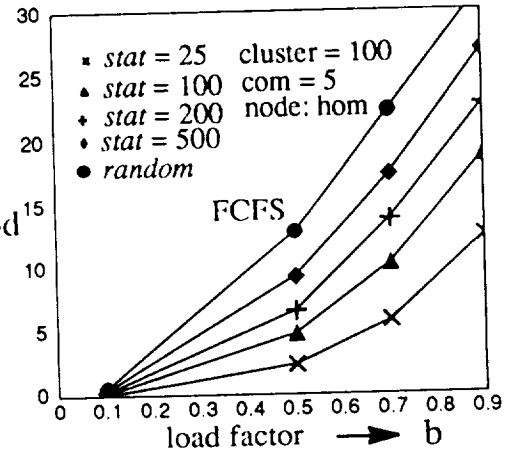
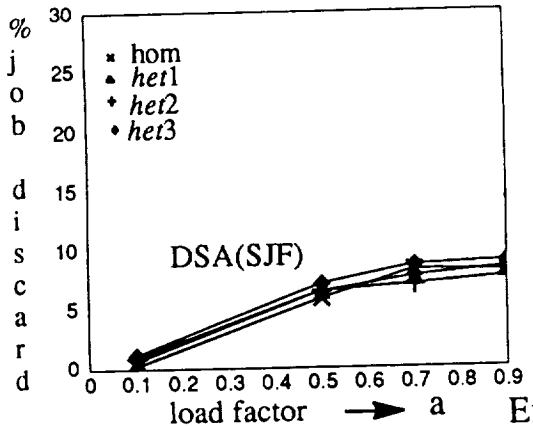


Figure 4:
Effect of updation period



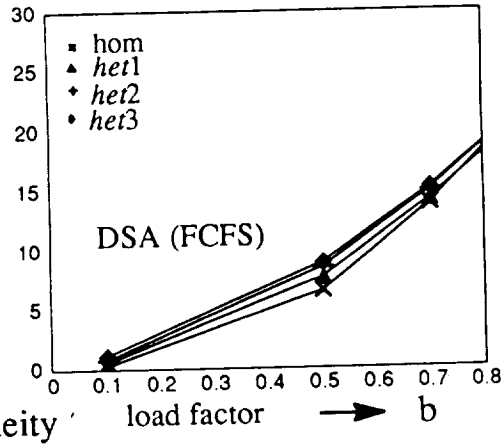
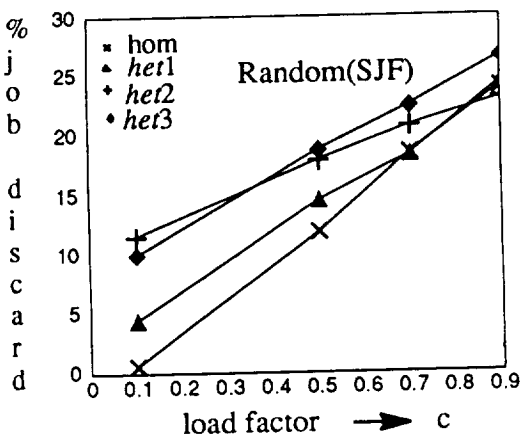


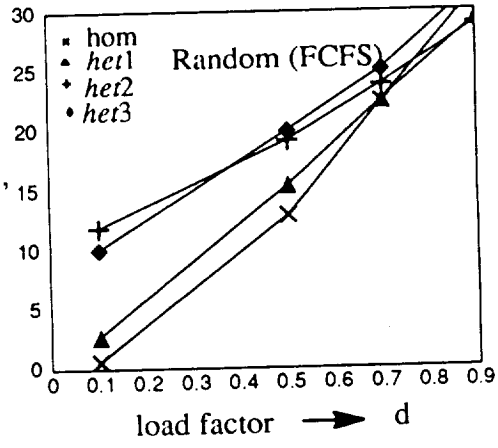
Figure 5:
Effect of Node Heterogeneity



het1 : <50,1.0> , <25,1.5> , <25,0.5>

het2 : <50,0.5> , <50,1.5>

het3 : <20,0.25> , <20,0.5> , <20,1.0>, <20,1.5>, <20 ,1.75>



A DISTRIBUTED SCHEDULING ALGORITHM FOR HETEROGENEOUS REAL-TIME SYSTEMS

Osman ZeinElDine Mohamed El-Toweissy Ravi Mukkamala
Department of Computer Science, Old Dominion University
Norfolk, Virginia 23529.

Abstract

Much of the previous work on load balancing and scheduling in distributed environments was concerned with homogeneous systems and homogeneous loads. Several of the results indicate that random policies are as effective as other more complex load allocation policies. In this paper, we investigate the effects of heterogeneity on scheduling algorithms for hard-real time systems. We propose a distributed scheduler specifically to handle heterogeneities in both nodes and node traffic. The performance of this algorithm is measured in terms of the percentage of jobs discarded. While a random task allocation is very sensitive to heterogeneities, our algorithm is shown to be robust to such non-uniformities in system components and load.

1. Introduction

Meeting deadline is of utmost importance for jobs in hard real-time systems. In these systems, when a job cannot be executed to completion prior to its deadline, it is either considered as having a zero value or as a discarded job with possible disastrous side-effects. Scheduling algorithms are assigned the responsibility of deriving job schedules so as to maximize the number of jobs that meet their deadline. Most of the literature in real-time systems deals with periodic deterministic tasks which may be prescheduled and executed cyclically [1,2,5,6]. The aperiodically arriving random tasks with deadlines have not been thoroughly investigated [3]. In addition, much of the studies in this area assume systems dedicated to real-time applications and working at extremely small loads (10% or less).

In a distributed real-time system, jobs with deadlines are received at each of the nodes in the system. Each node is generally capable of executing a job completely. These jobs are scheduled for execution at the nodes by a scheduler. A distributed scheduler is a distributed algorithm with cooperating agents at each node. Basically, the agents cooperate through exchange of local load information. The decision for scheduling a job, however, is taken by a local agent. Much of the current work in distributed scheduling assumes identical scheduling algorithms, homogeneous processing capabilities, and identical request arrival patterns at all nodes across the distributed system [7].

In this paper, we are interested in investigating the effects of heterogeneity and aperiodicity in distributed real-time systems on the performance of the overall system. We have designed a distributed scheduler specifically aimed at tolerating heterogeneities

in distributed systems. Our algorithm is based on a tree-structured scheduler where the leaves of the tree represent the processing nodes of the distributed system, and the intermediate nodes represent controlling or server nodes. The server node is a guardian for nodes below it in the tree. The leaf nodes attempt to keep its guardian node informed of their load status. When a leaf node cannot meet the deadline of an arriving job, it transfers the job to its guardian (or server). The guardian then sends this job either to one of its other child nodes or to its guardian. We measure the performance of our algorithm in terms of percentage of discarded jobs. (A job may be discarded either by a leaf node or by one of the servers.) Since random scheduling is often hailed to be as effective as some other algorithms with more intelligence, we compare the performance of our algorithm with a random scheduler [4]. Even though a random algorithm is often effective in a homogeneous environment, our investigations found it to be unsuitable for heterogeneous environments.

This paper is organized as follows. Section 2 presents the model of the system adopted in this paper. Section 3 describes the proposed scheduling algorithm. Section 4 summarizes the results obtained from simulations of our algorithm and a random scheduler algorithm. Finally, Section 5 presents some conclusions from this study and proposes some future work.

2. The System Model

For the purposes of scheduling, the distributed system is modeled as a tree of nodes and is shown in Figure 1. (The choice of the hierarchical structure is influenced by our scheduling algorithm which can handle a system with hundreds of nodes. The choice of three levels in this paper is for ease of illustration.) The nodes at the lowest level (level 0) are the *processing* nodes while the nodes at the higher levels represent guardians or *servers*. A processing node is responsible for executing arriving jobs when they meet some specified criteria. The processing nodes are grouped into *clusters*, and each cluster is assigned a unique server. When a server receives a job, it tries to either redirect that job to a processing node within its cluster or to its guardian. It is to be noted that this hierarchical structure could be logical (i.e., some of the processing nodes may themselves assume the role of the servers).

In summary, there are four components in the system model: jobs, processing nodes, servers, and the communication subsystem. A job is characterized by its arrival time, execution time, deadline, and priority (if any). The specifications of a processing node include its speed factor, scheduling policy, external arrival rate (of jobs), and job mix (due to heterogeneity). A server is modeled by its speed and its node assignment policy. Finally, the communication subsystem is represented by the speeds of transmission and distances between different nodes (processing and servers) in the distributed system.

3. Proposed Scheduling Algorithm

We describe the algorithm in terms of the participation of the three major components: the processing node, the sever at level 1, and the server at level 2. The major execution steps involved at these three components are summarized in Figure 2.

3.1 General

First let us consider the actions at the processing node. When a job arrives (either from an external user or from the server) at a *processing node*, it is tested at the gateway. The local scheduling algorithm at the node decides whether or not to execute this job locally. This decision is based on pending jobs in the local queue (which are already guaranteed to be executed within their deadlines), the requirements of the new job, and the scheduling policies (e.g., FCFS, SJF, SDF, SSF etc. [8]). In case the local scheduler decides to execute it locally, it will insert it in the local processing queue, and thereby guaranteeing to execute the new job within its deadline. By definition, no other jobs already in the local queue will miss their deadlines after the new addition. In case the local scheduler cannot make a guarantee to the new job, it either sends the job to its server (if there is a possibility of completion), or discard the job (if there is no such chance of completion).

Let us now consider the actions at level-1 server. Upon arriving at a *server*, a job enters the server queue. First, the server attempts to choose a candidate processing node (within its cluster) that is most likely to meet the deadline of this job. This decision is based on the latest information provided by the processing node to the server regarding its status. This information includes the scheduling algorithm, current load and other status information at each of the processing nodes in its cluster. (The choice of the information content as well as its currency are critical for efficient scheduling. For lack of space, we omit this discussion here.) When more than one candidate node is available, a random selection is carried out among these nodes. (We found a substantial difference in performance between choosing the first candidate and the random selection.) If a server cannot find a candidate node for executing the job, it forwards the job to the level-2 server.

The level-2 server (or *top level server*) maintains information about all level-1 servers. Each server sends an abstract form of its information to the level-2 server. Once again, the information content as well as its currency are crucial for the performance of the algorithm. This server redirects an arriving job to one of the level-1 servers. The choice of candidate servers is dependent on the ability of these servers to redirect a job to one of the processing nodes in their cluster to meet the deadline of the job.

The rule for discarding a job is very simple. At any time, a job may be discarded if the processing node or the server at which the job exists finds that if the job is sent elsewhere it would never be executed before its deadline. This may be due to the jobs already in the processing queue, and/or the communication delay for navigation along the hierarchy.

3.2 Information Abstraction at Different Levels

The auxiliary information (about the load status) maintained at a processing node or a server is crucial to the performance of the scheduling algorithm. Besides the information content, its structure and its maintenance will dictate its utility and overhead on the system. To maximize the benefit and minimize the overhead, every level is assigned

its own information structure. The information at the servers is periodically updated by nodes at the lower level. (The time interval for propagating the information to the servers is a parameter of the system.)

The jobs waiting to be executed at a processing node are classified according to the *local* scheduling algorithm (e.g., the classification would be on based priority if a priority scheduler is used.). Due to this dependency on the scheduling algorithm, we allow each processing node to choose its own local classification. Typically, the following attributes are maintained for each class:

- the average response time; (used for performance statistics)
- the likely end of execution (time) of the last job, including the one currently being served;
- the minimum slack among the jobs currently in the processing queue.

In addition, depending on the scheduling policy, we may have some other attributes.

The server maintains a copy of the information available at each of its child nodes including the scheduling policy. Since the information at a child node is dependent on the local scheduling policy, the server node should be capable of maintaining different types of data. Using this information, the server should be able to decide which processing nodes are eligible for executing a job and meet its deadline.

The information at level 2 server consists of an abstraction of the information available at each of the level-1 servers. Since each level-1 server may contain nodes with heterogeneous scheduling policies, level-2 server abstracts its information based on scheduling policies for each server. Thus, for a FCFS scheduling policy, it will contain abstracted status information from each of its child servers. This is repeated for other scheduling policies. Thus, the information at this level does not represent information at a processing node; rather it is a summary of information of a group of nodes in a cluster with the same scheduling policy.

4. Results

In order to evaluate the effectiveness of our scheduling algorithm, we have built a simulator and made a number of runs. The results presented in this paper concentrate on the sensitivity of our algorithm to four different parameters: the cluster size, the communication delay (between nodes), the frequency of propagation of load information (between levels), and the node heterogeneity. For lack of space, we have omitted other results such as the scheduler's sensitivity to heterogeneity in scheduling algorithms, heterogeneity in loads, and the effects of information structures. Accordingly, all the results reported here assume:

- FCFS scheduling policy at every node,
- the total number of processing nodes is 100,
- equal load at all nodes,
- communication delay between any nodes is the same.

The performance of the scheduler is measured in terms of the percentage of jobs discarded by the algorithm (at levels 0,1, 2). The rate of arrivals of jobs and their processing requirements are combinedly represented through a load factor. This load factor refers to the load on the overall system. Our load consists of jobs from three types of execution time constraints. The first type are short jobs with average execution of 10 units of time and with a slack of 25 units. (The actual values for a job are derived from an exponential distribution.) The second type of jobs have an average execution time of 50 units and a slack of 35 units. Long jobs have average execution times of 100 units and slacks of 300 units. In all our experiments, all these types have equal contribution to the overall load factor.

We now discuss our observations regarding the characteristics of the distributed scheduler in terms of the four selected parameters. In order to isolate the effect of one factor from others, the choice of parameters is made judiciously. For example, in studying the effects of cluster size (Figure 3), the updation period is chosen to be a medium value of 200 (stat=200), the communication delay is chosen to be small (com=5), and the nodes are assumed to be homogeneous (node: hom). Similarly, while studying the effects of the updation period (Figure 4), the cluster size is chosen to be 100 (cluster =100). Similar choices are made in the study of other two factors.

4.1 Effect of Cluster Size

Cluster size indicates the number of processing nodes being assigned to a level-1 server. In our study, we have considered three cluster sizes: 100, 50, and 10. A cluster of 100 nodes indicates a centralized server structure where all the processing nodes are under one level-1 server. In this case, level-2 server is absent. Similarly, in the case of cluster of 50 nodes, there are two level-1 servers, and one level-2 server. For 10-node cluster, we have 10 level-1 servers. In addition, we consider a completely decentralized case represented by the *random* policy. In this case, each processing node acts as its own server and randomly selects a destination node to execute a job which it cannot locally guarantee. We make the following observations (Figure 3).

- Both cluster sizes of 100 and 50 nodes have identical effect on performance.
- With cluster size of 10, the percentage of discarded jobs has increased. This difference is apparent at high loads.
- The random policy results in a significantly higher percentage of jobs being discarded.

From here, we conclude that our algorithm is robust to variations in cluster size. In addition, its performance is significantly superior to a random policy.

4.2 Effect of The Frequency of Updations

As is the case for all distributed algorithms, the currency of information at a node about the rest of the system plays a major role in performance. Hence, if the state of processing nodes vary rapidly, then the frequency of status information exchanges

between the levels should also be high. In order to determine the sensitivity of the proposed algorithm to the period of updating statistics at the servers, we experimented with four time periods: 25, 100, 200 and 500 units. (These time units are the same as the execution time units of jobs.) The results are summarized in Figure 4. From here we make the following observations.

- Our algorithm is extremely sensitive to changes in period of information exchanges between servers and processing nodes.
- Even in the worst case of 500 units, the performance of our algorithm is significantly better than the random policy.

4.3 Effect of Communication Delay

Since processing nodes send jobs that cannot be executed locally to a server, communication delay is a major factor in reducing the number of jobs discarded due to deadline limitations. Here, we present results for four values of of communication delay: 0, 5, 10 and 20 units. (These units are the same as the execution time units of jobs.) The results are summarized in Figure 5. The communication delay of zero represents a closely coupled system with insignificant time of inter-node or inter-process communication. A higher communication delay implies lower slack for jobs that cannot be executed locally. It may be observed that

- When the communication delay is 0, 5, or 10, the number of discarded jobs with our algorithm (DSA) is relatively small and independent of this delay. In all these cases, the percentage of discarded jobs with DSA is much smaller than with random policy.
- When communication delay is 20, however, the percentage of discarded jobs is much higher. In fact, in this case the random policy has a better performance than our algorithm.
- The performance difference between random policy and our algorithm reduces with the increase in communication delay. When the communication delay is higher, this difference has vanished, and in fact the random policy has displayed better performance.

We attribute our observation to the selection of slacks for the input jobs. If a shortest job could not be executed at the processing node at which it originated, it has to go through two hops of communication (node to server, server to node), resulting in twice the delay for a single hop. Since the maximum value of the slack for jobs with short execution time has been taken to be 25 units of time (in our runs), any communication delay above 12 units will result in a non-local job being discarded with certainty. Thus, even though our algorithm is robust to variations in communication delay, there is an inherent relationship between the slack of an incoming job and the communication delay.

4.4 Effect of Node Heterogeneity

As mentioned in the introduction, a number of studies claim that sending a job randomly over the network would be almost as good as using a complex load balancing algorithm [4]. We conjecture that this claim is only valid under homogeneous nodes assumption and jobs with no time constraints. Since one of our major objectives has been to test this claim for jobs with time constraints over a set of heterogeneous nodes, experiments have been conducted under four conditions. For each of these conditions, we derive results for our algorithm (DSA) as well as the random policy. The results are summarized in Figure 6. The homogeneous case (denoted by *hom*) represents a system with all 100 nodes having the same unit speeds. (Since a job is only distinguished by its processing time requirements, we have considered only speed heterogeneities.) The three heterogeneous cases are represented by *het1, het2, het3*. The heterogeneities are described through a set of <# of nodes, speed factor> pairs. For example, *het1* relates to a system with 50 nodes with a speed factor of 0.5 and 50 nodes with a speed factor of 1.5. Thus the average speed of a processing node is still 1.0, which is the same as the homogeneous case. The other two heterogeneous cases may be similarly explained. Among the cases considered, the degree of heterogeneity is maximum for *het3*. From our results it may be observed that:

- With our algorithm, even though the increase in degree of heterogeneity resulted in an increase of discarded jobs, the increase is not so significant. Hence, our algorithm appears to be robust to node heterogeneities.
- The performance of the random policy is extremely sensitive to the node heterogeneity. As the heterogeneity is increased, the number of discarded jobs is also significantly increased.

With the increase in node heterogeneity, the number of nodes with slow speed also increase. Thus, using a random policy, if a slow speed node is selected randomly, then the job is more likely to be discarded. In our algorithm, since the server is aware of the heterogeneities, it can suitably avoid a low speed node when necessary. Even in this case, there is a tendency for high-speed nodes to be overloaded and low speed nodes to be under loaded. Hence, the difference in performance.

5. Conclusions

In this paper, a distributed algorithm has been proposed to help schedule jobs with time constraints over a network of heterogeneous nodes, each of which could have its own processing speed and job-scheduling policy. Several parametric studies have been conducted. From the results obtained it can be concluded that:

- the algorithm has a large improvement over the random selection in terms of the percentage of discarded jobs;
- the performance of the algorithm tends to be invariant with respect to node-speed heterogeneity;

- the algorithm efficiently utilizes the available information; this is evident from the sensitivity of our algorithm to the periodicity of update information.
- the performance of the algorithm is robust to variations in in the cluster size.

In summary, our algorithm is robust to several heterogeneities commonly observed in distributed systems. Our other results (not presented here) also indicate the robustness of this algorithm to heterogeneities in scheduling algorithms at local nodes. In future, we propose to extend this work to investigate the sensitivity of our algorithm to other heterogeneities in distributed systems. We also propose to test its viability in a non-real time system where response time or throughput is the performance measure.

ACKNOWLEDGEMENT

This research was sponsored in part by the NASA Langley Research Center under contracts NAG-1-1114 and NAG-1-1154.

References

[1] S. R. Biyabani, J.A. Stankovic, and K. Ramamritham, "The integration of deadline and criticalness in hard real-time scheduling," *Proc. Real-time Systems Symposium*, pp. 152-160, December 1988.

[2] J.-Y. Chuang and J.W.S. Liu, "Algorithms for scheduling periodic jobs to minimize average error," *Proc. Real-time Systems Symposium*, pp. 142-151, December 1988.

[3] D. W. Craig and C.M. Woodside, "The rejection rate for tasks with random arrivals, deadlines, and preemptive scheduling," *IEEE Trans. Software Engineering*, Vol. 16, No. 10, pp. 1198-1208, Oct. 1990.

[4] D. L. Eager, E.D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. Software Engineering*, Vol. SE-12, No. 5, pp. 662-675, May 1986.

[5] R. Rajkumar, L. Sha, and J.P. Lehoczky, "Real-time synchronization protocols for multiprocessors," *Proc. Real-time Systems Symposium*, pp. 259-269, December 1988.

[6] K.G. Shin, C.M. Krishna, and Y.-H. Lee, "Optimal resource control in periodic real-time environments," *Proc. Real-time Systems Symposium*, pp. 33-41, December 1988.

[7] J. Stankovic and K. Ramamritham, "Evaluation of a bidding algorithm for hard real-time distributed systems," *IEEE Trans. Computers*, Vol. C-34, No. 12, pp. 1130-1143, Dec. 1986.

[8] W. Zhao, K. Ramamritham, and J. Stankovic, "Scheduling tasks with resource requirements in hard-real time systems," *IEEE Trans. Software Engineering*, Vol. SE-13, No. 5, pp. 564-577, May 1987.

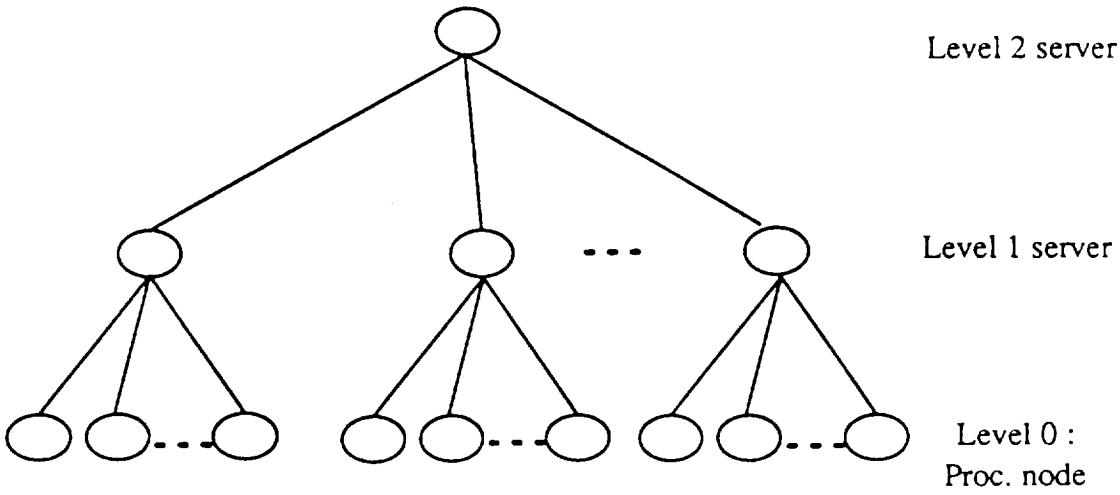


Figure 1 : System Model

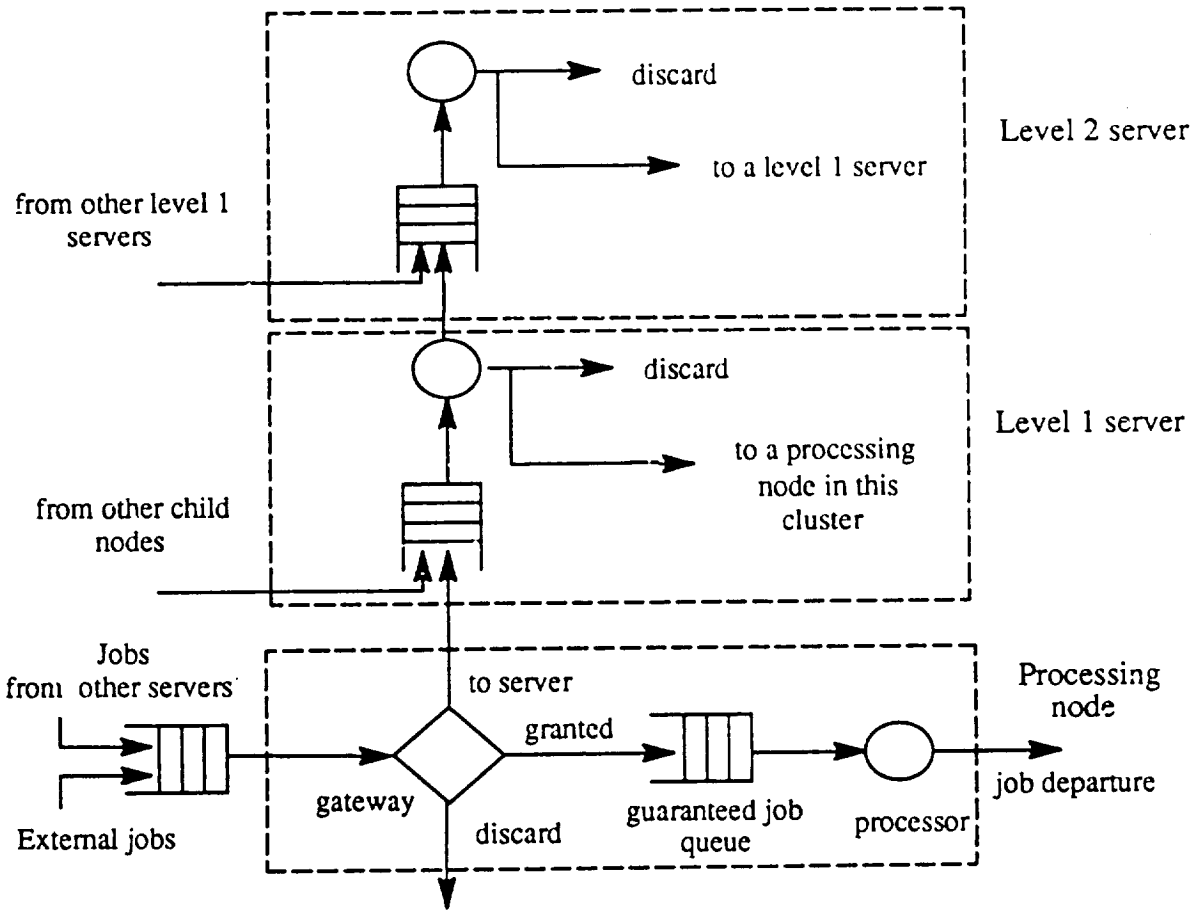


Figure 2 : Flow diagram of The Algorithm

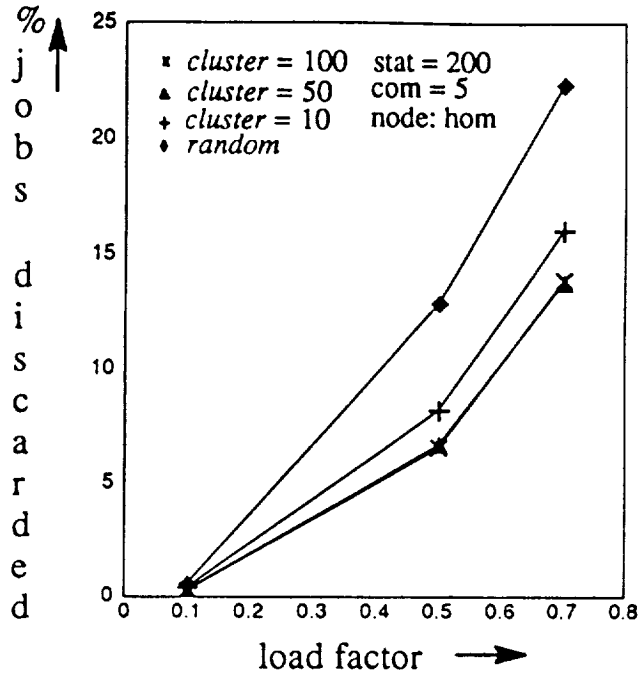


Figure 3: Effect of cluster size

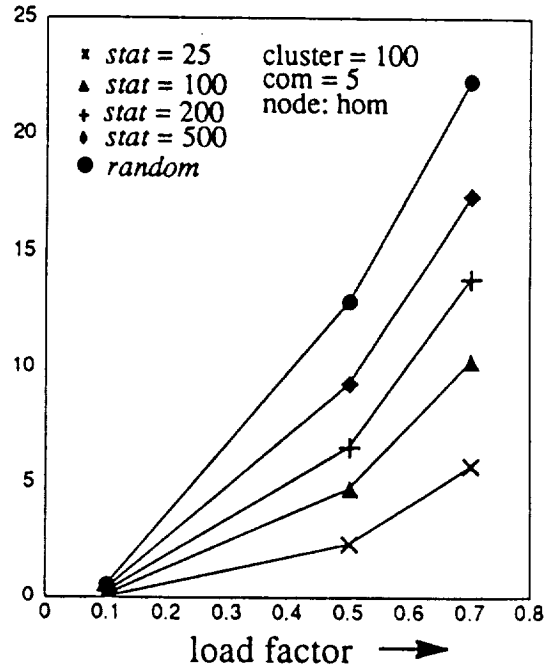
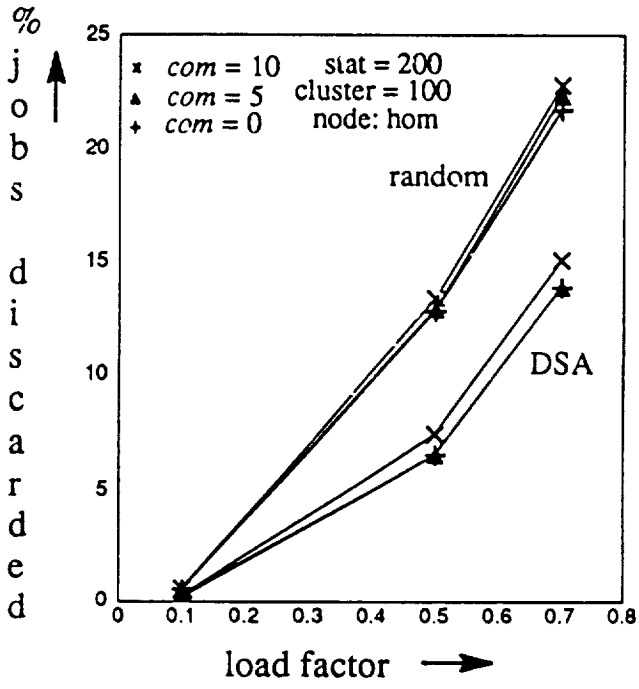


Figure 4: Effect of updation period



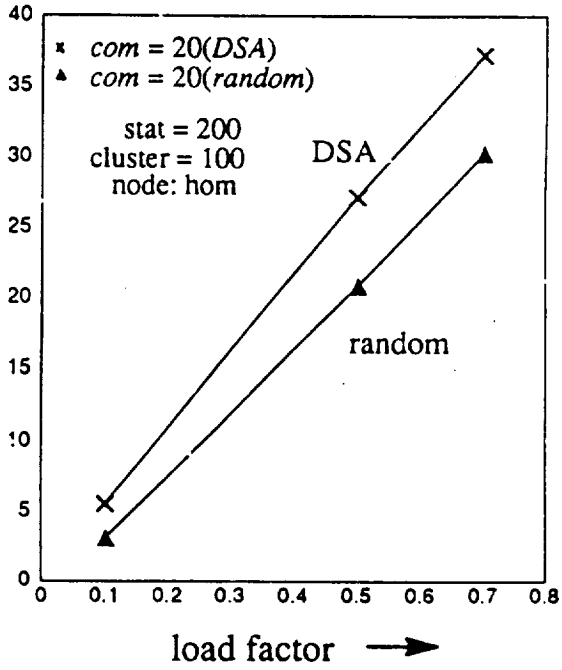


Figure 5: Effect of communication delay

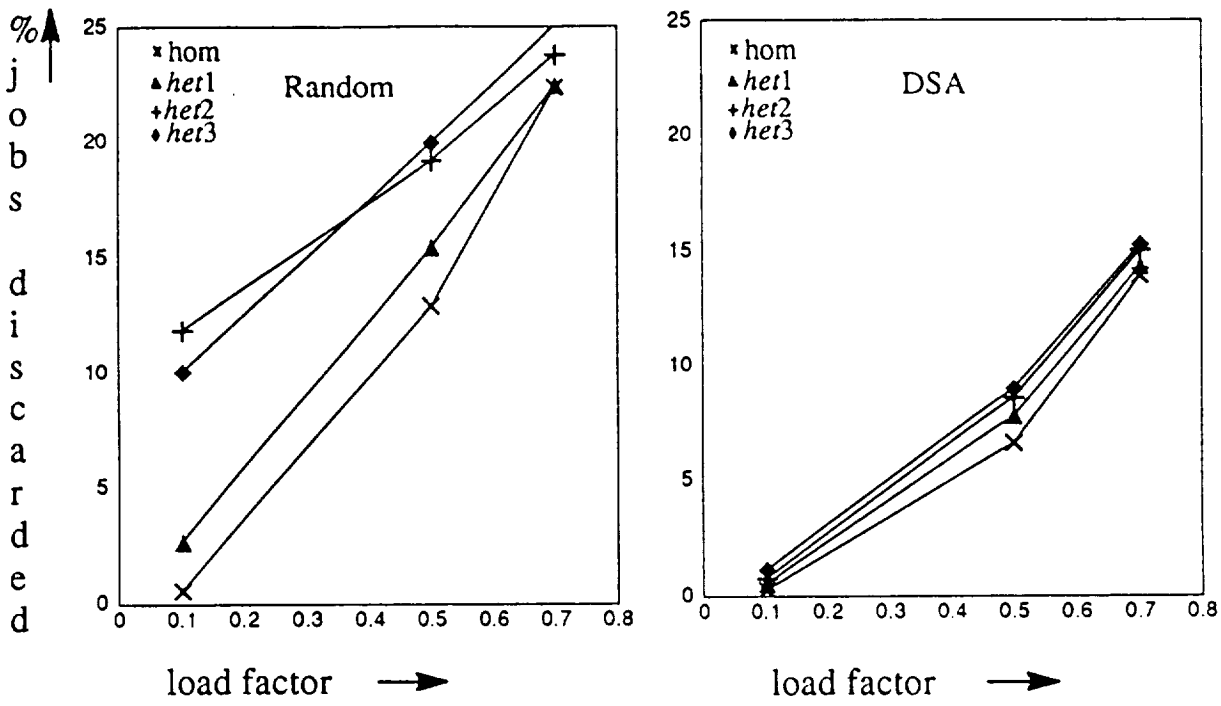


Figure 6: Effect of Node Heterogeneity

het1 : <50,1.0> , <25,1.5> , <25,0.5>
het2 : <50,0.5> , <50,1.5>
het3 : <20,0.25> , <20,0.5> , <20,1.0> , <20,1.5> , <20,1.75>

IEEE PROCEEDINGS OF THE

SOUTHEASTCON '91

Volume 2

91CH2998-3

NASA

Performance Analysis of Static Locking in Replicated Distributed Database Systems

Yinghong Kuang
Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.

Abstract

Data replications and transaction deadlocks can severely affect the performance of distributed database systems. Many current evaluation techniques ignore these aspects, because it is difficult to evaluate through analysis and time-consuming to evaluate through simulation. In this paper, we use a technique that combines simulation and analysis to closely illustrate the impact of deadlock and evaluate performance of replicated distributed database with both shared and exclusive locks.

1. Introduction.

A distributed database system (DDS) is a collection of cooperating nodes each containing a set of data items. A user transaction can enter such a system at any of these nodes. The receiving node, often referred to as the *coordinating* node, undertakes the task of locating the nodes that contain the data items required by a transaction.

In order to maintain database consistency and correctness in the presence of concurrent transactions, several concurrency control protocols have been proposed [1]. Of these, the most commonly used are time-stamping and locking protocols. Locking protocols have been widely used in both commercial and research environments. In static locking, prior to start of execution, a transaction needs to acquire either a shared-lock (for read operations) or an exclusive lock (for update operations) on each of the relevant data items.

Data replication is used to improve the performance of local transactions and the availability of databases. In replicated databases, one data item may have more than one copy in the system. Replica control algorithms are used to maintain the consistency among these copies. One of these is the read-one/write-all protocol. With this protocol an exclusive lock need to acquire an exclusive lock from every copy of the data item . For a shared lock to succeed, any one copy of the data item has to be share locked. When transactions with conflicting lock requests are initiated concurrently, they could be possibly blocked due to a deadlock.

There are two major ways to evaluate the performance of distributed systems: simulation and analysis. Simulation is a conceptually tractable technique, but requires large computation time. On the other hand, analysis is computationally faster but may not be tractable for all problems. In [4], Shyu and Li proposed an elegant analysis model to evaluate the response time and throughput of transactions in a non-replicated DDS. Assuming *exclusive* locking (i.e., only write operations), they model the queue of lock requests at an object as an M/M/1 queue [3]. This results in a closed-form for the waiting time distribution at a node, expressed in terms of the average rates of arrivals of requests and the average lock-holding time. With shared lock and replications added into the picture, it is very difficult to have a close model for it. Because of the limitations of simulation and analysis, we develop a technique that combines simulation and analysis.

This paper is organized as the follows. In Section 2, we describe the model used in our performance evaluation. In Section 3, we propose an evaluation technique. In Section 4, we illustrate the results. Finally, Section 5 has the conclusions.

2. Model

Our model has the following parameters:

- There are n nodes.
- There are d data items in a DDS.
- A data item may be located at exactly c number of nodes. The dc data copies are uniformly distributed across the n nodes.
- Each transaction accesses k data items.
- r is the read ratio. So among k data items to be accessed, rk are accessed only for read operations, and the rest are for read-write operations. Due to the read-one/write-all replica control policy, a transaction must procure rk shared locks for rk read operations and $(1-r)kc$ exclusive locks for the $(1-r)k$ read-write operations.
- Each data item is equally likely to be accessed by a transaction.
- Transaction arrivals into the system is a Poisson process with rate λ.
- The communication delay between any two nodes is exponentially distributed with mean $\bar{t}$.
- The average execution time of a transaction, once the locks are obtained, is s.
- The deadlock mechanism is invoked every τ seconds.
- After an abortion of a transaction, it takes an average of ω seconds for this transaction to be restarted.
- μ is the service rate of transactions.
- b is the lock-holding time.
- λc is the arrival rate at each data copy.

[1]This research was supported in part by the NASA Langley Research Center under contracts NAG-1-1114 and NAG-1-1154.

CH2998-3/91/0000-0698$01.00©1991IEEE

3. Performance Evaluation Technique

Our technique consists of two stages. In the first stage, the average transaction response time and throughput are calculated by ignoring the deadlock. This is an iterative step involving simulation and analysis. In the second stage, the probabilities of transaction conflicts and deadlocks are computed by probability models. These probabilities are used, in turn, to compute the response time and throughput in the presence of deadlocks.

Stage 1:

Initially, we assume that there are no lock conflicts between transactions. Each transaction has to procure rk shared lock on data copies and $(1-r)kc$ exclusive locks on data copies. When a transaction has got all the lock grants from these data objects, it can go ahead with execution.

This procedure is summarized in the following 6 steps.

1. Initialize lock-holding time(b) to be $1/\mu$.
2. Given the total rate of transaction arrival(λ), the shared lock ratio(r), the number of data items(d), the number of data items required by each transaction(k) and the number of replications(c), derive the arrival rate at each data copy(λc).
3. With the arrival rate at each data copy(λc), the average lock-holding time(b), and the transmission time(t) we can simulate the queue at a data copy to arrive wait-time(w) distribution. With this distribution we can calculate the response time of transactions.
4. With the average service time of transactions($1/\mu$), and the transmission time, we can derive a new lock-holding time(b').
5. Set b to this new lock-holding time b'.
6. If the old and new lock-holding time are sufficiently close, stop the iteration. Otherwise, go back to step 3.

At the end of stage 1 the response time without the consideration of transaction deadlocks is obtained.

Stage 2:

This stage considers transaction conflicts and computes the deadlock probability. Here the probabilities of transaction deadlock and restart are computed. These are then used to compute response time and throughput in the presence of deadlocks.

Assume there are two transactions T1 and T2. Let RS, WS be the read and write sets of transactions respectively.

1. Let fs_i be the probability that the readset of T1 has i data items overlapping with the writeset of T2, i.e. $|RS(T1) \cap WS(T2)| = i$.
2. Let fe_{ij} be the probability that given $|RS(T1) \cap WS(T2)| = i$, the writeset of T1 has j data items overlaping with the readset and writeset of T2, i.e. the probability that $|WS(T1) \cap (RS(T2) \cup WS(T2))| = j$.

Clearly,

$$fs_i = \frac{\binom{k-rk}{i}\binom{d-k+rk}{rk-i}}{\binom{d}{rk}} \quad (1)$$

$$fe_{ij} = \frac{\binom{k-i}{j}\binom{d-rk-k+i}{k-rk-j}}{\binom{d-rk}{k-rk}} \quad (2)$$

It can also be noted that $fs_i fe_{ij}$ is the probability that:
|Read-set(T1)∩Write-set(T2)|=i
∧ |Write-set(T1)∩(Write-set(T2)∪Read-set(T2))|=j.
If PW_{ij} is the probability that T1 waits for T2,

$$PW_{ij} = p1 + p2 - p1 * p2 \quad (3)$$

$$p1 = 1 - [1 - (1/2)^c]^i \quad (4)$$

$$p2 = (1 - (1/2)^{cj}) \quad (5)$$

where $p1$ is the probability that T1 waits for T2 for shared locks in readset
and $p2$ is the probability that T1 waits for T2 for exclusive locks in writeset.
Probability that T1 waits for T2 is now given by

$$Pw = \sum_{i=0}^{\min(r,k-r)} \sum_{j=0}^{\min(k-r,k-i)} fs_i fe_{ij} PW_{ij} \quad (6)$$

With this probability of waiting and the formulas in [4] we can calculate the probability of a transaction deadlock, the probability of a transaction restart and the probability of a transaction to be blocked by other transactions. And with these probabilities and the time between deadlock detection(τ), we can calculate the response time with consideration of deadlock. (Details are omitted here.)

4. Results

Using this technique, we obtained a number of interesting results that illustrate the effect of deadlocks and number of replications on database performance. These are summarized in Figures 1-5. We make the following observations.

- Transaction response times are quite sensitive to the ratio of shared locks (Figure 1 and 2). Here, we compare the response times when deadlocks are ignored (DI, computed in Stage 1) with those obtained when deadlocks are considered (DC, computed in Stage 2). The effect of deadlocks is more predominant at higher transaction loads and with smaller values of r. When $r = 2/3$, the effect of deadlocks is not significant on response time.
- If we compare Figure 1 and 2 with Figure 3 and 4, it can be observed that the increase in replications results in the larger response time when read ratio is smaller than 1/3.
- Fig. 5 shows the response times with different replication numbers. Here we can see that with both cases when read ratio is 2/3 and 1/3, the response time increases as the number of replications increases. But with read ratio equals 1/3, the increasing rate is much smaller than that with read ratio equals 2/3.

5. Conclusions

In [4], Shyu and Li presented an elegant technique to evaluate the performance of distributed database systems in the presence of deadlocks. Their technique assumed only exclusive locks and thus representing the worst-case effects of deadlocks.

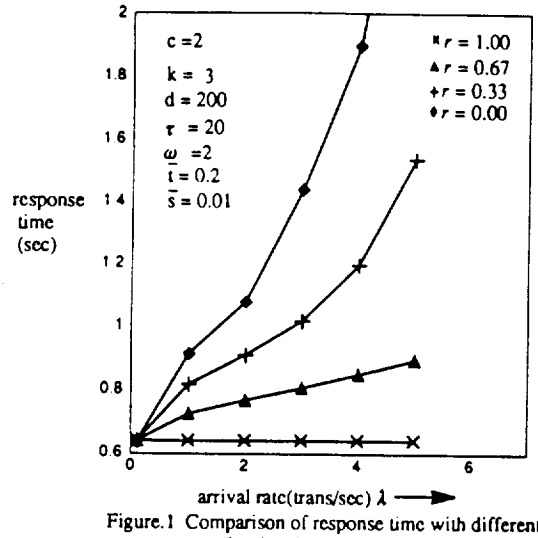


Figure.1 Comparison of response time with different read ratio when deadlock is ignored.

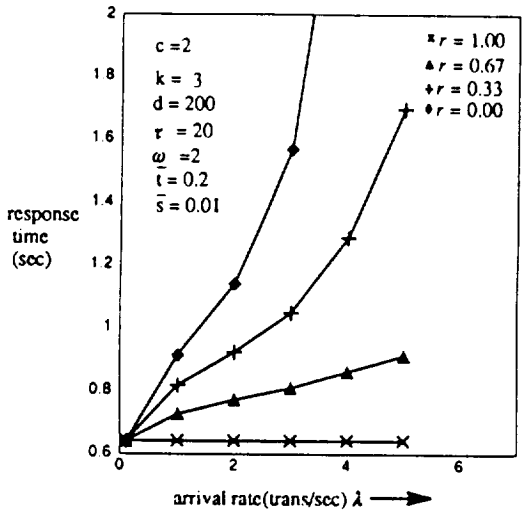


Figure.2 Comparison of response time with different read ratio when deadlock is considered.

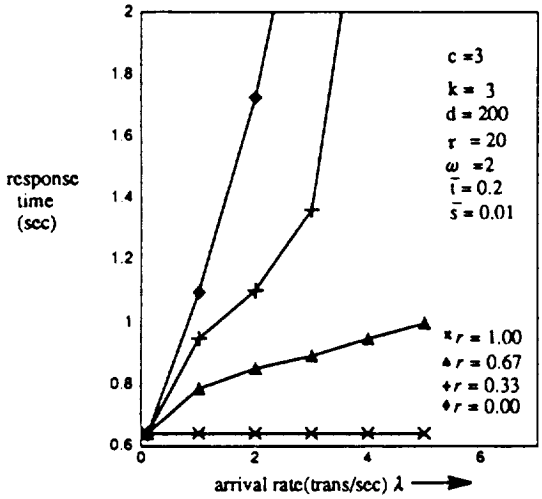


Figure.3 Comparison of response time with different read ratio when deadlock is ignored.

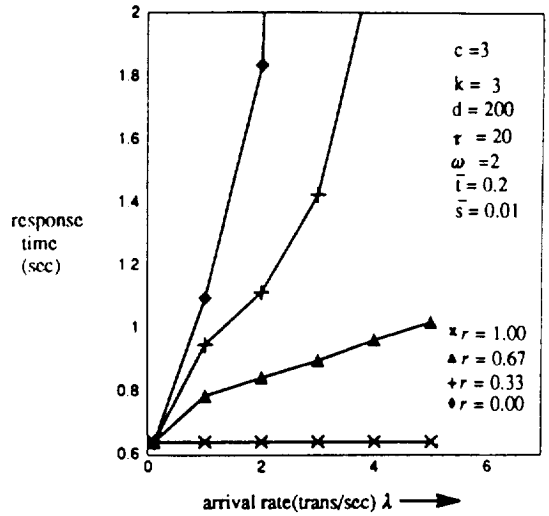


Figure.4 Comparison of response time with different read ratio when deadlock is ignored.

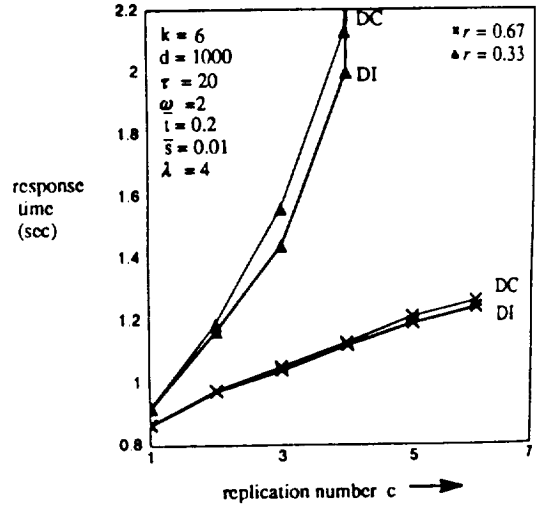


Figure. 5 Comparison of response time with different read ratio with and without deadlock.

DC: Deadlock Considered.
DI: Deadlock Ignored.

In this paper, we have extended their technique to combine [illegible] ulation and analysis. And with this extended technique we also [illegible] both shared and exclusive locking and also replications in our model. We evaluated the the effect of number of data items, the number of data items accessed by each transaction, the ratio of read operations on transaction response time and the number of replications. These results show the importance of considering both shared and exclusive lock requests, the deadlock probabilities as well as the number of replications of database in response time evaluations.

References

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

[2] A. Hac, "A decomposition solution to a queueing network model of a distributed file system with dynamic locking," *IEEE Trans. Software Eng.*, vol. SE-12, no. 4, pp. 521-530, Apr. 1986.

[3] L. Kleinrock, *Queueing Systems, Vol. I*, New York: Wiley-Interscience, 1975.

[4] S.-C. Shyu and V. O. K. Li, "Performance analysis of static locking in distributed database systems," *IEEE Trans. Computers*, vol. 39, no. 6, pp. 741-751, June 1990.

[5] M. Singhal and A. K. Agrawala, "Performance analysis of an algorithm for concurrency control in replicated database systems," *Proc. ACM SIGMETRICS Conf. Measurement Modeling Comput. Syst.*, 1986, pp. 159-169.

[6] Y. C. Tay, R. Suri, and N. Goodman, "A mean value performance model for locking in databases: The no-waiting case," *J. ACM*, vol. 32, no. 3, pp. 618-651, July 1985.

Proceedings of the 1990 Winter Simulation Conference
Osman Balci, Randall P. Sadowski, Richard E. Nance (eds.)

EFFECTS OF DISTRIBUTED DATABASE MODELING ON EVALUATION OF TRANSACTION ROLLBACKS

Ravi Mukkamala

Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529-0162.

ABSTRACT

Data distribution, degree of data replication, and transaction access patterns are key factors in determining the performance of distributed database systems. In order to simplify the evaluation of performance measures, database designers and researchers tend to make simplistic assumptions about the system. In this paper, we investigate the effect of modeling assumptions on the evaluation of one such measure, the number of transaction rollbacks, in a partitioned distributed database system. We develop six probabilistic models and develop expressions for the number of rollbacks under each of these models. Essentially, the models differ in terms of the available system information. The analytical results so obtained are compared to results from simulation. From here, we conclude that most of the probabilistic models yield overly conservative estimates of the number of rollbacks. The effect of transaction commutativity on system throughput is also grossly undermined when such models are employed.

1. INTRODUCTION

A distributed database system is a collection of cooperating nodes each containing a set of data items (In this paper, the basic unit of access in a database is referred to as a data item.). A user transaction can enter such a system at any of these nodes. The receiving node, sometimes referred to as the *coordinating* or *initiating* node, undertakes the task of locating the nodes that contain the data items required by a transaction.

A partitioning of a distributed database (DDB) occurs when the nodes in the network split into groups of communicating nodes due to node or communication link failures. The nodes in each group can communicate with each other, but no node in one group is able to communicate with nodes in other groups. We refer to each such group as a *partition*. The algorithms which allow a partitioned DDB to continue functioning generally fall into one of two classes [Davidson et al. 1985]. Those in the first class take a *pessimistic* approach and process only those transactions in a partition which do not conflict with transactions in other partitions, assuring mutual consistency of data when partitions are reunited. The algorithms in the second class allow every group of nodes in a partitioned DDB to perform new updates. Since this may result in independent updates to items in different partitions, conflicts among transactions are bound to occur, and the databases of the partitions will clearly diverge. Therefore, they require a strategy for conflict detection and resolution. Usually, rollbacks are used as a means for preserving consistency; conflicting transactions are rolled back when partitions are reunited. Since coordinating the undoing of transactions is a very difficult task, these methods are called *optimistic* since they are useful primarily in a situation where the number of items in a particular database is large and the probability of conflicts among transactions is small.

In general, determining if a transaction that successfully executed in a partition is rolled back at the time the database is merged depends on a number of factors. Data items in the read-set and the write-set of the transaction, the distribution of these data items among the other partitions, access patterns of transactions in other partitions, data dependencies among the transactions, and semantic relation (if any) between these transactions are some examples of these factors. Exact evaluation of rollback probability for all transactions in a database (and hence the evaluation of the number of rolled back transactions) generally involves both analysis and simulation, and requires large execution times [Davidson 1982; Davidson 1984]. To overcome the computational complexities of evaluation, designers and researchers generally resort to approximation techniques [Davidson 1982; Davidson 1986; Wright 1983a; Wright 1983b]. These techniques reduce the computation time by making simplifying assumptions to represent the underlying distributed system. The time complexity of the resulting techniques greatly depends on the assumed model as well as evaluation techniques.

In this paper we are interested in determining the effect of the distributed database models on the computational complexity and accuracy of the rollback statistics in a partitioned database.

The balance of this paper is outlined as follows. Section 2 formally defines the problem under consideration. In Section 3, we discuss the data distribution, replication, and transaction modeling. Section 4 derives the rollback statistics for one distribution model. In Section 5, we compare the analysis methods for six models and simulation method for one model based on computational complexity, space complexity, and accuracy of the measure. Finally, in Section 6, we summarize the obtained results.

2. PROBLEM DESCRIPTION

Even though a transaction T_1 in partition P_1 may be rolled back (at merging time) by another transaction T_2 in partition P_2 due to a number of reasons, the following two cases are found to be the major contributors [Davidson 1982].

i. $P_1 \neq P_2$, and there is at least one data item which is updated by both T_1 and T_2. This is referred to as a *write-write* conflict.

ii. $P_1 = P_2$, T_2 is rolled back, and it is a *dependency parent* of T_1 (i.e., T_1 has read at least one data item updated by T_2, and T_2 occurs prior to T_1 in the serialization sequence).

The above discussion on reasons for rollback only considers the syntax of transactions (i.e. read- and write-sets) and does not recognize any semantic relation between them. To be more specific, let us consider transactions T_1 and T_2 executed in two different partitions P_1 and P_2 respectively. Let us also assume that the intersection between the write-sets of T_1 and T_2 is non-empty. Clearly, by the above definition, there is a write-write conflict and one of the two transactions has to be rolled back. However, if T_1 and T_2 commute with each other, then there is no need to rollback either of the transactions at the time of partition merge [Garcia-Molina 1983; Jajodia and Speckman 1985; Jajodia and Mukkamala 1990]. Instead, T_1 needs to be executed in P_2 and T_2 needs to be executed in P_1. The analysis in this paper take this property into account.

In order to compute the number of rollbacks, it is also necessary to define some ordering ($O(P)$) on the partitions. For example, if T_1 and T_2 correspond to case (i) above, and do not commute, it is necessary to determine which of these two are rolled back at the time of merging. Partition ordering resolves this ambiguity by the following rule: Whenever two conflicting but non-commuting transactions are executed in two different partitions, then the transaction executed in the lower order partition is rolled back.

Since a transaction may be rolled back due to either (i) or (ii), we classify the rollbacks into two classes: Class 1 and Class 2 respectively. The problem of estimating the number of rollbacks at the time of partition merging in a partially replicated distributed database system may be formulated as follows.

Given the following parameters, determine the number of rolled back transactions in class 1 (R_1) and class 2 (R_2).

- n, the number of nodes in the database;
- d, the number of data items in the database;
- p, the number of partitions in the distributed system (prior to merge);
- t, the number of transaction types;
- GD, the global data directory that contains the location of each of the d data items; the GD matrix has d rows and n columns, each of which is either 0 or 1;
- NS_k, the set of nodes in partition k, $\forall k = 1, 2, \ldots, p$;
- RS_j, the read-set of transaction type j, $j = 1, 2, \ldots, t$;
- WS_j, the write-set of transaction type j, $j = 1, 2, \ldots, t$;
- N_{jk}, the number of transactions of type j received in partition k (prior to merge), $j = 1, 2, \ldots, t$, $k = 1, 2, \ldots, p$.
- CM, the commutativity matrix that defines transaction commutativity. If $CM_{j_1 j_2} = true$ then transaction types j_1 and j_2 commute. Otherwise they do not commute.

The average number of total rollbacks is now expressed as $R = R_1 + R_2$.

3. MODEL DESCRIPTION

As stated in the introduction, the primary objective of this paper is to investigate the effect of data distribution, replication, and transaction models on estimation of the number of rollbacks in a distributed database system.

To describe a data distribution-transaction model, we characterize it with three orthogonal parameters:

1. Degree of data item replication (or the number of copies).
2. Distribution of data item copies.
3. Transaction characterization

We now discuss each of these parameters in detail.

For simplicity, several analysis techniques assume that each data item has the same number of copies (or degree of replication) in the database system [Coffman et al. 1981]. Some other techniques characterize the degree of replication of a database by the average degree of replication of data items in that database [Davidson 1986]. Others treat the degree of replication of each data item independently.

Some designers and analysts assume some specific allocation schemes for data item (or group) copies (e.g., [Mukkamala 1987]). Assuming complete knowledge of data copy distribution (GD) is one such assumption. Depending on the type of allocation, such assumptions may simplify the performance analysis. Others assume that each data item copy is randomly distributed among the nodes in the distributed system [Davidson 1986].

Many database analysts characterize a transaction by the size of its read-set and its write-set. Since different transactions may have different sizes, these are either classified based on the sizes, or an average read-set size and average write-set size are used to represent a transaction. Others, however, classify transactions based on the data items that they access (and not necessarily on their size). In this case, transaction types are identified with their expected sizes and the group of data items from which these are accessed. An extreme example is a case where each transaction in the system is identified completely by its read-set and its write-set.

With these three parameters, we can describe a number of models. Due to the limited space, we chose to present the results for six of these models in this paper.

We chose the following six models based on their applicability in the current literature, and their close resemblance to practical systems. In all these models, the rate of arrival of transactions at each of the nodes is assumed to be completely known a priori. We also assume complete knowledge of the partitions (i.e. which nodes are in which partitions) in all the models.

Model 1: Among the six chosen models, this has the maximum information about data distribution, replication, and transactions in the system. It captures the following information.

- *Replication:* Data replication is specified for each data item.
- *Data distribution:* The distribution of data items among the nodes in the system is represented as a distribution matrix (as described in Section 2).
- *Transactions:* All distinct transactions executed in a system are represented by their read-sets and write-sets. Thus, for a given transaction, the model knows which data items are read, and which data items are updated. The commutativity information is also completely known and is expressed as a matrix (as described in Section 2).

Model 2: This model reduces the number of transactions by combining them into a set of transaction types based on commutativity, commonalities in data access patterns, etc. Since the transactions are now grouped, some of the individual characteristics of transactions (e.g. the exact read-set and writes-set) are lost. This model has the following information.

- *Replication:* Average degree of replication is specified at the system level.
- *Data distribution:* Since the read- and write-set information is not retained for each transaction type, the data distribution information is also summarized in terms of average data items. It is assumed that the data copies are allocated randomly to the nodes in the system.
- *Transactions:* A transaction type is represented by its read-set size, write-set size, and the number of data items from which selection for read and write is made. Since two transaction types might access the same data item, it also stores this overlap information for every pair of transaction types. The commutativity information is stored for each pair of transaction types.

Model 3: This model further reduce the transaction types by grouping them based only on commutativity characteristics. No consideration is given to commonalities in data access pattern or differing read-set and write-set sizes. It has the following information.

- *Replication:* Average degree of replication is specified at the system level.
- *Data distribution:* As in model 2, it is assumed that the data copies are allocated randomly to the nodes in the system.
- *Transactions:* A transaction type is represented by the average read-set size and average write-set size. The commutativity information is stored for all pairs of transaction types.

Model 4: This model classifies transactions into three types: read-only, read-write, and others. Read-only trans-

actions commute among themselves. Read-write transactions neither commute among themselves nor commute with others. The others class corresponds to update transactions that may or may not commute with transactions in their own class. This fact is represented by a commute probability assigned to it.

- *Replication:* Average degree of replication is specified at the system level.
- *Data distribution:* As in model 2, it is assumed that the data copies are allocated randomly to the nodes in the system.
- *Transactions:* Read-only class is represented by average read-set size. The read-write class is represented by average read-set and write-set sizes. The others class is represented by the average read-set size, average write-set size and the probability of commutation.

Model 5: This model reduces the transactions to two classes: read-only and read-write. Read-only transactions commute among themselves. The read-write transactions corresponds to update transactions that may or may not commute with transactions in their own class. This fact is represented by a commute probability assigned to it.

- *Replication:* Average degree of replication is specified at the system level.
- *Data distribution:* As in model 2, it is assumed that the data copies are allocated randomly to the nodes in the system.
- *Transactions:* Read-only class is represented by average read-set size. The read-write class is represented by average read-set and write-set sizes, and the probability of commutation.

Model 6: This model identifies read-only transactions and other update transactions. But these two types have the same average read-set size. Update transactions may or may not commute with other update transactions.

- *Replication:* Average degree of replication is specified at the system level.
- *Data distribution:* As in model 2, it is assumed that the data copies are allocated randomly to the nodes in the system.
- *Transactions:* The read-set size of a transaction is denoted by its average. For update transactions, we also associate an average write-set size and the probability of commutation.

Among these, model 1 is very general, and assumes complete information of data distribution (GD), replication, and transactions. Other models assume only partial (or average) information about data distribution and replication. Model 1 has the most information and model 6 has the least.

4. COMPUTATION OF THE AVERAGES

Several approaches offer potential for computing the average number of rollbacks for a given system environment; the most prominent methods are simulation and probabilistic analysis.

Using simulation, one can generate the data distribution matrix (GD) based on the data distribution and replication policies of the given model. Similarly, one can generate different transactions (of different types) that can be received at the nodes in the network. Since the partition information is completely specified, by searching the relevant columns of the GD matrix, it is possible to determine whether a given transaction has been successfully executed in a given partition. Once all the successful transactions have been identified, and their data dependencies are identified, it is possible to identify the transactions that need to be rolled back at the time of merging. The generation and evaluation process may have to be repeated enough number of times to get the required confidence in the final result.

Probabilistic analysis is especially useful when interest is confined to deriving the average behavior of a system from a given model. Generally, it requires less computation time. In this paper, we present detailed analysis for model 6, and a summary of the analysis for models 1-5.

4.1 Derivations for Model 6

This model considers only two transaction types: read-only (Type 1) and read-write (Type 2). Both have the same average read-set size of r. A read-write transaction updates w of the data items that it reads. N_{1k} and N_{2k} represent the rate of arrival of types 1 and 2 respectively at partition k. The average degree of replication of a data item is given as c. The system has n nodes and d data items. The probability that two read-write transaction commute is m.

Let us consider an arbitrary transaction T_1 received at one of the nodes in partition k with n_k nodes. Since the copies of a data item are randomly distributed among the n nodes, the probability that a single data item is accessible in partition k is given by

$$\alpha_k = 1 - \frac{\binom{n-n_k}{c}}{\binom{n}{c}} \quad (1)$$

Since each data item is independently allocated, the expected number of data items available in this partition is $d\alpha_k$. Similarly, since T_1 accesses r data items (on the average), the probability that it will be successfully executed is α_k^r. From here, the number of successful transactions in k is estimated as $\alpha_k^r N_{1k}$ and $\alpha_k^r N_{2k}$ for types 1 and respectively.

In computing the probability of rollback of T_1 due to case (i), we are only interested in transactions that update a data item in the write-set of T_1 and not commuting with T_1. The probability that a given data item (updated by T_1) is not updated in another partition k' by a non-commuting transaction (with respect to T_1) is given by

$$\beta_{k'} = \left(1 - \frac{w}{d\alpha_{k'}}\right)^{(1-m)\alpha_{k'}^r N_{2k'}} \quad (2)$$

Given that a data item is available in k, probability that it is not available in k' is given as

$$\gamma(k,k') = \frac{\binom{n-n_{k'}}{c} - \binom{n-n_k-n_{k'}}{c}}{\alpha_k \binom{n}{c}} \quad (3)$$

From here, the probability that a data item available in k is not updated any other transaction in higher order partitions is given as

$$\delta_k = \prod_{\forall k', O(k')>O(k)} [\gamma(k,k') + (1-\gamma(k,k'))\beta_{k'}] \quad (4)$$

The probability that transaction T_1 is not in write-write conflict with any other non-commuting transaction of higher-order partitions is now given as

$$\mu_k = \frac{\binom{d\alpha_k\delta_k}{w}}{\binom{d\alpha_k}{w}} \quad (5)$$

From here, the number of transactions rolled back due to category (i) may be expressed as $R_1 = \sum_{k=1}^{p}(1-\mu_k)\alpha_k^r N_{2k}$

To compute the rollbacks of category (ii), we need to determine the probability that T_1 is rolled back due to the rollback of a dependency parent in the same partition. If T_2 is a read-write transaction in partition k, then the probability that T_1 depends on T_2 (i.e. read-write conflict) is given by:

$$\lambda_k = 1 - \frac{\binom{do_k - w}{r}}{\binom{do_k}{r}} \quad (6)$$

The probability that T_i is not rolled back due to the roll back of any of its dependency parents is now given by:

$$\chi_k = \sum_{i=1}^{\alpha_k^r N_k} \frac{(\lambda_k \mu_k + 1 - \lambda_k)^{ui}}{\alpha_k^r N_k} \quad (7)$$

where $N_k = N_{1k} + N_{2k}$ and $u = N_{2k}/(N_{1k} + N_{2k})$.

The total number of rolled back transactions due to category (ii) is now estimated as $R_2 = \sum_{k=1}^{p}(1 - \chi_k)\alpha_k^r(N_{1k} + \mu_k N_{2k})$. The total number of rolled back transactions is $R = R_1 + R_2$.

5. COMPARISON OF THE MODELS

As mentioned in the introduction, the main objective of this paper is to determine the effect of data distribution, replication, and transaction models on the estimation of rollbacks. To achieve this, we evaluate the desired measure using six different data distribution and replication models. The comparison of these evaluations is based on computational time, storage requirement, and the average values obtained.

Due to the limited space, we could not present the detailed derivations for the average values for models 2-6. The final expressions, however, are presented in [Mukkamala 1990].

5.1 Computational Complexity

We now analyze each of the evaluation methods (for models 1-6) for their computational complexity.

- In model 1, all t transactions are completely specified, and the data distribution matrix is also known. To determine if a transaction is successful, we need to the scan the distribution matrix. Similarly, determining if a transaction in a lower order partition is to be rolled back due to a write-write conflict with a transaction of higher order partition requires comparison of write-sets of the two transactions. Determining if a transaction needs to be rolled back due to the rollback of a dependency parent also requires a search. All this requires $O(ndt + p^2t^2 + pt^2N)$, where t is the number of transaction types and N is the maximum number of transactions executed in a partition prior to the merge.
- Models 2-6 have a similar computation structure. The number of transaction types (t) is high for model 2 and low for model 6. Each of these models require $O(p^2t^2c + pt^2N)$ time. As before, t is the number of transaction types and N is the maximum number of transactions executed in a partition prior to the merge.

Thus, model 1 is the most complex (computationally) and model 6 is the least complex.

5.2 Space Complexity

We now discuss the space complexity of the six evaluation methods:

- Model 1 requires $O(dn)$ to store the data distribution matrix, $O(n)$ to store the partition information, $O(dt)$ to store the data access information, and $O(nt)$ to store the transaction arrival information. It also requires $O(t^2)$ to store the commutativity information. Thus, it requires $O(dn + dt + nt + t^2)$ space to store model information.
- Models 4-6 require similar information: $O(t)$ to store the average size of read- and write- sets of transaction types, $O(nt)$ for transaction arrival, $O(n)$ for partition information, and $O(t)$ for commute information. Thus they require $O(nt)$ space.
- Model 3, in addition to the space required by models 4-6, also requires $O(t^2)$ for commutativity matrix. Thus it requires $O(nt + t^2)$ space.
- Model 2, in addition to the space required by model 3, also requires t^2 space to store the data overalp information. Thus, it requires $O(nt + t^2)$ storage.

Thus, model 1 has the largest storage requirement and model 6 has the least.

5.3 Evaluation of the Averages

In order to compare the effect of each of these models on the evaluation of the average rollbacks, we have run a number of experiments. In addition to the analytical evaluations for models 1-6, we have also run simulations with Model 1. The results from these runs are summarized in Tables 1-7. Basically these tables describe the number of transactions successfully executed before partition merge (*Before Merge*), number of rollbacks due to class 1 (R_1), rollbacks due to class 2 (R_2), and transactions considered to be successful at the completion of merge (*After Merge*). Obviously, the last term is computed from the earlier three terms. In all these tables, the total number of transaction arrivals into the system during partitioning is taken to be 65000. Also, each node is assumed to receive equal share of the incoming transactions.

- Table 1 summarizes the effect of number of partitions as measured with Models 1-6. Here, it is assumed that each of the data items in the system has exactly $c = 3$ copies. The other assumptions in models 1-6 are as follows:
  1. Model 1 considers 130 transaction types in the system. Each is described by its read- and write-sets and whether it commutes with the other transactions. 90 of the 130 are read-only transactions. The rest of the 40 are read-write. Among the read-write, 15 commute with each other, another 10 commute with each other, and the rest of the 15 do not commute at all. The simulation run takes the same inputs but evaluates the averages by simulation.
  2. Model 2 maps the 130 transaction types into 4 classes. To make the comparisons simple, the above four classes (90+15+10+15) are taken as four types. The data overlap is computed from the information provided in model 1.
  3. Model 3, to facilitate comparison of results, considers the above 4 classes. This model, however, does not capture the data overlap information.
  4. Model 4 considers three types: read-only, read-write that commute among themselves with some probability, and read-write that do not commute at all.
  5. Model 5 considers read-only transactions with read-set size of 3 and read-write transactions with read-set size of 6. Read-write transactions commute with a given probability.
  6. Model 6 only considers the average read-set size (computed as 4 in our case), the portion of read-write transactions (=45/130), and the average write-set size for a read-write (= 2). Probability that any two transactions commute is taken to be 0.4.

From Table 1 it may be observed that:

- The analytical results from analysis of Model 1 is a close approximation of the ones from simulation.
- The evaluation of number of successful transactions prior to the merge is well approximated by all the models. Model 6 deviated the most.
- The difference in estimations of R_1 and R_2 is significant across the models. Model 1 is closest to the

simulation. Model 6 has the worst accuracy. Model 5, surprisingly, is somewhat better than Models 2,3,4, and 6.

- The estimation of R_2 from models 2-6 is about 50 times of the estimation from Model 1. The estimations from Model 1 and the simulation are quite close. From here, we can see that, Models 2-6 yield overly conservative estimates of the number of rollbacks at the time of partition merge. While Model 1 estimated the rollbacks as 1200, Model 2-6 have approximated them as about 13000.
- This difference in estimations seems to exist even when the number of partitions is increased.

- Table 2 summarizes the effect of number of copies on the evaluation accuracies of the models. It may be observed that
  - The difference between evaluations from Model 1 and the others is significant at low ($c = 3$) as well as high ($c = 8$) values of c. Clearly, the difference is more significant at high degrees of replication.
  - The case $p_1 = 4, p_2 = 6, c = 8$ corresponds to a case where each of the 500 data items is available in both the partitions. This is also evident from the fact that all the 65000 input transactions are successful prior to the merge.
  - The results from the analysis and simulation of Model 1 are close to those from simulation.
- Table 3 shows the effect of increasing the number of nodes from 10 (in Table 1) to 20. For large values of n, all the six models result in good approximations of successful transactions prior to merge. The differences in estimations of R_1 and R_2 still persist.
- Table 4 compares models 5 and 6. While model 6 only retains average read-set size information for any transaction, model 6 keeps this information for read-only and read-write transactions separately. This additional information enabled model 5 to arrive at better approximations for R_1 and R_2. In addition, the effect of commutativity on R_1 and R_2 is not evident until $m \geq 0.99$. This is counterintuitive. The simplistic nature of the models is the real cause of this observation. Thus, even though these models have resulted in conservative estimates of R_1 and R_2, we can't draw any positive conclusions about the effect of commutativity on the system throughput.
- The comments that were made about the conservative nature of the estimates from models 5 and 6 also applies to model 2. These results are summarized in Table 5. Even though this model has much more system information than models 5 and 6, the results (R_1 and R_2) are not very different. However, the effect of commutativity can now be seen at $m \geq 0.95$.
- Having observed that the effect of commutativity is almost lost for smaller values of m in models 2-6, we will now look at its effect with model 1. These results are summarized in Table 6. Even at small values of m, the effect of commutativity on the throughput is evident. In addition, it increases with m. This observation holds at both small and large values of c.
- In Table 7, we summarize the effect of variations in number of copies. In Tables 1-6, we assumed that each data item has exactly the same number of copies. This is more relevant to Model 1. Thus we only consider this model in determining the effect of copy variations on evaluation of R_1 and R_2. As shown in this table, the effect is significant. As the variation in number of copies is increased, the number of successful transactions prior to merge decreases. Hence, the number of conflicts are also reduced. This results in a reduction of R_1 and R_2. AS long as the variations are not very significant, the differences are also not significant.

6. CONCLUSIONS

In this paper, we have introduced the problem of estimating the number of rollbacks in a partitioned distributed database system. We have also introduced the concept of transaction commutativity and described its effect on transaction rollbacks. For this purpose, the data distribution, replication, and transaction characterization aspects of distributed database systems have been modeled with three parameters. We have investigated the effect of six distinct models on the evaluation of the chosen metric. These investigations have resulted in some very interesting observations. This study involved developing analytical equations for the averages, and evaluating them for a range of parameters. We also used simulation for one of these models. Due to lack of space, we could not present all the obtained results in this paper. In this section, we will summarize our conclusions from these investigations.

We now summarize these conclusions.

- Random data models that assume only average information about the system result in very conservative estimates of system throughput. One has to be very cautious in interpreting these results.
- Adding more system information does not necessarily lead to better approximations. In this paper, the system information is increased from model 6 to model 2. Even though this increases the computational complexity, it does not result in any significant improvement in the estimation of number of rollbacks.
- Model 1 represents a specific system. Here, we define the transactions completely. Thus it is closer to a real-life situation. Results (analytical or simulation) obtained from this model represent actual behavior of the specified system. However, results obtained from such a model are too specific, and can't be extended for other systems.
- Transaction commutativity appears to significantly reduce transaction rollbacks in a partitioned distributed database system. This fact is only evident from the analysis of model 1. On the other hand, when we look at models 2-6, it is possible to conclude that commutativity is not helpful unless it is very very high. Thus, conclusions from model 1 and models 2-6 appear to be contradictory. Since models 3-6 assume average transactions that can randomly select any data item to read (or write), the evaluations from these models are likely to predict higher conflicts and hence more rollbacks. The benefits due to commutativity seem to disappear in the average behavior. Model 1, on the other hand, describes a specific system, and hence can accurately compute the rollbacks. It is also able to predict the benefits due to commutativity more accurately.
- The distribution of number of copies seems to affect the evaluations significantly. Thus, accurate modeling of this distribution is vital to evaluation of rollbacks.

In addition to developing several system models and evaluation techniques for these models, this paper has one significant contribution to the modeling, simulation, and performance analysis community.

> If an abstract system model with average information is employed to evaluate the effectiveness of a new technique or a new concept, then we should only expect conservative estimates of the effects. In other words, if the results from the average models are positive, then accept the results. If these are negative, then repeat the analysis with a less abstracted model. Concepts/techniques that are not appropriate for an average system may still be applicable for some specific systems.

Table 1. Effect of Number of Partitions on Rollbacks

Model #	$p_1 = 4, p_2 = 6, c = 3$ Before Merge	R_1	R_2	After Merge	$p_1 = 4, p_2 = p_3 = 3, c = 3$ Before Merge	R_1	R_2	After Merge
Sim.	50200	1000	205	48995	31450	0	0	31450
1	50200	1000	199	49001	31450	0	0	31450
2	48315	3597	10322	34397	27069	3460	8945	14664
3	48315	3464	10194	34657	27069	2798	9410	14861
4	48618	3667	10243	34708	27657	3255	9444	14958
5	47276	2679	10238	34360	24207	1507	9106	13594
6	46593	3852	8570	34171	22356	2937	6673	12747

Table 2. Effect of Number of Copies on Rollbacks

Model #	$p_1 = 4, p_2 = 6, c = 2$ Before Merge	R_1	R_2	After Merge	$p_1 = 4, p_2 = 6, c = 8$ Before Merge	R_1	R_2	After Merge
Sim.	34600	200	15	34385	65000	4000	4970	56030
1	34600	200	0	34400	65000	4000	4981	56019
2	31069	1998	5119	23952	65000	8000	17777	39223
3	31069	1601	5334	24134	65000	8000	17786	39214
4	31595	1798	5420	24377	65000	8000	17786	39214
5	23203	1568	2326	19309	65000	8000	17875	39125
6	27138	3413	1701	22024	65000	8000	17860	39140

Table 3. Effect of Number of Nodes on Rollbacks

Model #	$p_1 = 10, p_2 = 10, c = 5$ Before Merge	R_1	R_2	After Merge	$p_1 = 10, p_2 = 10, c = 12$ Before Merge	R_1	R_2	After Merge
Sim.	61250	4000	6240	51010	65000	5000	6231	53769
1	61250	4000	6231	51019	65000	5000	6231	53769
2	61024	9090	21183	30751	65000	10000	22277	32723
3	61024	8992	21286	30746	65000	10000	22286	32714
4	61100	9031	21326	30743	65000	10000	22286	32714
5	60968	9064	21292	30613	65000	10000	22375	32625
6	60876	9363	20936	30577	65000	10000	22360	32640

ACKNOWLEDGEMENT

This research was sponsored in part by the NASA Langley Research Center under contract NAG-1-1154.

REFERENCES

Coffman, E. G., E. Gelenbe, and B. Plateau (1981), "Optimization of Number of Copies in a Distributed Database," *IEEE Transactions on Software Engineering* 7, 1, 78-84.

Davidson, S.B. (1982), "An optimistic protocol for partitioned distributed database systems," Ph.D. thesis, Department of EECS, Princeton University.

Davidson, S.B. (1984), "Optimism and consistency in partitioned distributed database systems," *ACM Transactions on data* systems* 9, 3, 456-481.

Davidson, S.B., H. Garcia-Molina, and D. Skeen (1985), "Consistency in partitioned networks," *ACM Computing Surveys* 17, 3, 341-370.

Davidson, S.B. (1986), "Analyzing partition failure protocols," Technical Report MS-CIS-86-05, Department of Computer and Info. Sci., Univ. of Pennsylvania.

Garcia-Molina, H. (1983), "Using semantic knowledge for transaction processing in a distributed system," *ACM Trans. on Database Systems* 8, 2, 186-213.

Jajodia, S. and P. Speckman (1985), "Reduction of conflicts in partitioned databases," In *Proceedings of the 19th Annual Conference on Information Sciences and Systems*, 349-355.

Jajodia, S. and R. Mukkamala (1990), *Measuring the Effect of Commutative Transactions On Distributed Database Performance*, To appear in *Computer Journal*.

Mukkamala, R. (1987), "Design of Partially Replicated Distributed Database Systems," Technical Report 87-04, Department of Computer Science, University of Iowa.

Mukkamala, R. (1990), "Measuring the Effects of distributed database models on transaction rollback measures," Technical Report 90-38, Department of Computer Science, Old Dominion University.

Wright, D. D. (1983a), "Managing distributed databases in partitioned networks," Ph.D. thesis, Department of Computer Science, Cornell University, (also TR 83-572).

Wright, D. D. (1983b), "On merging partitioned databases," *ACM SIGMOD Record* 13, 4, 6-14.

Table 4. Effect of m on Rollbacks (Models 5 and 6: $p_1 = 4, p_2 = 6, c = 3$

	Model 5				Model 6			
m	Before Merge	R_1	R_2	After Merge	Before Merge	R_1	R_2	After Merge
0.00	47276	2679	10238	34360	46593	3852	8570	34171
0.50	47276	2679	10238	34360	46593	3852	8570	34171
0.80	47276	2679	10238	34360	46593	3852	8570	34171
0.90	47276	2679	10238	34360	46593	3848	8574	34171
0.95	47276	2678	10239	34360	46593	3774	8774	34175
0.99	47276	2208	10665	34403	46593	2182	10109	34301
1.00	46726	0	0	46726	46593	0	0	46593

Table 5. Effect of m on Rollbacks (Model 2: $p_1 = 4, p_2 = 6$)

	$c = 3$				$c = 8$			
m	Before Merge	R_1	R_2	After Merge	Before Merge	R_1	R_2	After Merge
0.0	48315	3597	10322	34397	65000	8000	17973	39027
0.27	48315	3597	10322	34397	65000	8000	17973	39027
0.40	48315	3597	10322	34397	65000	8000	17973	39027
0.77	48315	3597	10322	34397	65000	8000	17973	39027
0.95	48315	3205	10708	34402	65000	7660	18312	39028
0.99	48315	986	12882	34447	65000	4321	21642	39037
1.0	48315	0	0	48315	65000	0	0	65000

Table 6. Effect of m on Rollbacks (Model 1: $p_1 = 4, p_2 = 6$)

	$c = 3$				$c = 8$			
m	Before Merge	R_1	R_2	After Merge	Before Merge	R_1	R_2	After Merge
0.0	50200	4000	1199	45001	65000	8000	6379	50621
0.27	50200	1000	199	49001	65000	4000	4981	56019
0.40	50200	800	199	49201	65000	1800	2793	60407
0.77	50200	0	0	50200	65000	0	0	65000
1.0	50200	0	0	50200	65000	0	0	65000

Table 7. Effect of Variations in # of Copies on Rollbacks
(Model 1: $p_1 = 4, p_2 = 6, w/c : m = 0.27, wo/c : m = 0.0$)

$p_1 = 4, p_2 = 6, c = 3$					
Copy Distribution		Before Merge	R_1	R_2	After Merge
$d_3 = 500$	w/c	50200	1000	199	49001
	wo/c	50200	4000	1199	45001
$d_2 = d_4 = 100, d_3 = 300$	w/c	48300	1000	997	46303
	wo/c	48300	4200	1793	42307
$d_2 = d_3 = 167, d_4 = 166$	w/c	41400	200	0	41200
	wo/c	41400	2000	597	38803
$d_1 = d_2 = d_3 = d_4 = d_5 = 100$	w/c	40400	200	0	40200
	wo/c	40400	1600	797	38003
$d_1 = d_5 = 250$	w/c	28700	0	0	28700
	wo/c	28700	1200	199	27301

```
/* This program creates a menu and facilitates updates, inserts and
deletes of records in a database.  EMPP is an employee database and
the program assumes it to be already created with the following
fields:
EMPNO (employee number) of type numeric
ENAME (employee name) of type character
SAL (salary) of type numeric with provision for 2 places after
decimal
DEPTNO ( department number) of type numeric
JOB (job name) of type character
*/

#include <stdio.h>
#include <ctype.h>


EXEC SQL BEGIN DECLARE SECTION;
VARCHAR uid[80]; /* variable for user id */
VARCHAR pwd[20]; /* variable for password */

int  empno; /* host variable for primary key - employee number */
VARCHAR ename[15]; /* host variable for employee name */

int  deptno; /* department number */

VARCHAR job[15]; /* host variable for job */

int  sal; /* host variable for salary */

int l = 0; /* host variable to hold the length of the srting - a
value returned by the asks() function. */

int count; /* a variable to obtain number of records in the
database with the same primary key value */

int reply = 0; /* variable to obtain the whether a new value exists
*/

int choice = 0; /* variable defined to obtain value for the menu */


int code; /* variable to print to the ascii file to indicate wether
the record was updated (value=1), inserted (value=2) and deleted
(value=3) */

EXEC SQL END DECLARE SECTION;
EXEC SQL INCLUDE SQLCA;

FILE *fp;

main()

{
/* open ascii file in append mode */
```

```
   fp = fopen("outfile", "a");
/* give the login and password to logon to the database */
   strcpy(uid.arr,"rsp");
   uid.len = strlen(uid.arr);
   strcpy(pwd.arr,"prs");
   pwd.len = strlen(pwd.arr);

/* exit in case of an unauthorized accessor to the database */

   EXEC SQL WHENEVER SQLERROR GOTO errexit;
   EXEC SQL CONNECT :uid IDENTIFIED BY :pwd;
for (;;)
{ /* infinite loop begins */
 /* menu for selecting update, insert, and delete options */
     printf("\n \n 1.  Update a record \n");
     printf("\n \n 2.  Insert a record \n");
     printf("\n \n 3.  Delete a record \n");

     printf("\n \n Select an option 1/2/3 ? \n");

     choice = getche();
     if (choice == '1') goto update;
          else if (choice == '2') goto insert;
                    else if (choice =='3') goto delete;
                         else { printf("invalid selection");
                            exit(1);}
update: /* label for the update option */

  {
 /* To ensure that the employee with the given employee number
exists, before update could be made. */
     code = 1;
     printf(" \n count is %d \n", count);
     askn("Enter employee number to be updated: ", &empno);

/* using the COUNT supported by oracle, the number of records
having the desired employee number is assigned the variable count
*/

     EXEC SQL SELECT COUNT(EMPNO) INTO :count
     FROM EMPP
     WHERE EMPNO = :empno;
     printf("count is %d \n", count);

     if (count == 0)
     { printf("Employee with employee number %d does not exist \n",
empno);
       exit(1); }

/* retrieve the information from the database whose employee-number
has been requested for, and place the contents of the fields into
C variables for update purposes.  */

     EXEC SQL SELECT ENAME, SAL, DEPTNO, JOB
```

```
      INTO :ename, :sal, :deptno, :job
      FROM EMPP
      WHERE EMPNO = :empno;

/* displays the already existing value for employee name */
/* assign the new employee name if it should be updated */

      printf("ename is %s \n", ename.arr);
      printf("Do you want to update ENAME:(y/n)?");
      reply = getche();

      if (reply == 'n'){
         ename = ename;
         printf("\n ename is %s \n", ename.arr);}
      if (reply == 'y') {
         l = asks("\n enter employee name : ", ename.arr);
         printf("new ename is %s \n", ename.arr);}

/* displays the already existing value for job name */
/* assign the new job if it should be updated */

      printf("do you want to update job-name:(y/n)?");
      reply = getche();

      if (reply == 'n'){
         job = job;
         printf("\n job-name is %s \n", job.arr);}
      if (reply == 'y'){
         job.len = asks("\n enter employee's job :", job.arr);
         printf("new job-name is %s \n", job.arr);}

/* displays the already existing value for salary */
/* assign new salary if it should be updated */

   printf("Do you want to update salary:(y/n)" );
   reply = getche();

      if (reply == 'n'){
         sal = sal;
         printf("\n salary is %d \n", sal);}
      if (reply == 'y'){
         askn("\n enter employee's salary: ", &sal);
         printf("new salary is %d \n", sal);}

/* displays the already existing value for department number */
/* assign the new department number if it should be updated */

   printf("Do you want to update deptno :(y/n)");
   reply = getche();

      if (reply == 'n'){
         deptno = deptno;
         printf("\n deptno is %d \n", deptno);}
      if (reply == 'y'){
```

```
        askn("\n Enter employee dept  :   ",&deptno);
        printf("new deptno is %d \n", deptno);}

/* update the database with the new values */

     EXEC SQL UPDATE EMPP
     SET ENAME = :ename, SAL = :sal, DEPTNO = :deptno, JOB = :job
     WHERE EMPNO = :empno;

     printf("\n %s with employee number %d has been updated\n",
ename.arr,empno);

        fprintf(fp,"%10d %1d %15s", empno, code, ename.arr);
        fprintf(fp,"%6d %3d %4s\n",  sal, deptno, job.arr);

        printf("%10d %15s %6d", empno, ename.arr, sal);
        printf("%3d %4s\n", deptno, job.arr);
  }

insert: /* label for insertion of record based on the employee
number */

  {
     code = 2;

/* To prevent insertion of a record whose primary key is the same
as the primary key of an already existing record */

     askn("\n Enter employee number to be inserted:", &empno);
     EXEC SQL SELECT COUNT(EMPNO) INTO :count
     FROM EMPP
     WHERE EMPNO = :empno;

     printf("count is %d \n", count);

     if (count > 0){
       printf("Employee with %d employee number already exists \n",
empno);
       exit(1);}
     else { /* obtain values for various fields to be inserted */
          l = asks("Enter employee name : ", ename.arr);
          job.len = asks("Enter employee job :", job.arr);
          askn("Enter employee salary :", &sal);
          askn("Enter employee dept number :", deptno);

/* insert the values obtained into the database */

          EXEC SQL INSERT INTO EMPP(EMPNO,ENAME,JOB,SAL,DEPTNO)
          VALUES (:empno, :ename, :job, :sal, :deptno);

/* append the insert into the ascii file */

          fprintf(fp,"%10d %1d %15s ", empno, code, ename.arr);
          fprintf(fp,"%6d %3d %4s \n", sal, deptno, job.arr);
```

```
          printf("%10d %15s %6d", empno, ename.arr, sal);
          printf("%3d %4s", deptno,job.arr); }
  }

delete:  /* label for deletion of records based on employee number
*/

  {
  int code = 3;

/* obtain the employee number of the employee to be deleted */

  askn("Enter employee number to be deleted :", &empno);
  EXEC SQL SELECT COUNT(EMPNO) INTO :count
  FROM EMPP
  WHERE EMPNO = :empno;

  if (count > 0){ /* delete record if it exists */
     EXEC SQL DELETE FROM EMPP WHERE EMPNO = :empno;
     printf("Employee number %d deleted \n",empno);

     fprintf(fp, "%10d %1d\n", empno, code);}

  else {
     printf("Employee with number %d does not exist \n", empno);
     exit(1);}
  }

   EXEC SQL COMMIT WORK RELEASE; /* make the changes permanent */
   printf ("\n End of the C/ORACLE example program.\n");
   return;
     fclose(fp);

     EXEC SQL WHENEVER SQLERROR CONTINUE;
     EXEC SQL ROLLBACK WORK RELEASE; /* in case of inconsistency */

     return;

errexit:
   errrpt();

}
} /* infinite loop ends */

/* function takes the text to be printed and accepts a srting
variable from standard input and converts it into numeric - hence
is used to obtain values for numeric fields */

int askn(text,variable)
     char text[];
     int *variable;
     {
     char s[20];
     printf(text);
```

```
     fflush(stdout);
     if (gets(s) == (char *)0)
          return(EOF);

     *variable = atoi(s);
     return(1);
     }
/* function takes the text to be printed and prints it, accepts
string values for character variables and is thus used to obtain
values for fields of type character.  It returns the length of the
string value */

int asks(text,variable)
     char text[],variable[];
     {
     printf(text);
     fflush(stdout);
     return   (   gets(variable)   ==   (char   *)   0   ?   EOF   :
strlen(variable));
     }

errrpt()
     {
     printf("%.70s    (%d)\n",    sqlca.sqlerrm.sqlerrmc,
-sqlca.sqlcode);
     return(0);
     }
```

N91-25957

HETEROGENEOUS DISTRIBUTED DATABASES: A CASE STUDY

Tracy R. Stewart
Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia 23529.

1 INTRODUCTION

The purpose of this case study is to review alternatives for accessing distributed heterogeneous databases and propose a recommended solution. Our current study is limited to the Automated Information Systems Center at the Naval Sea Combat Systems Engineering Station at Norfolk, VA. This center maintains two databases located on Digital Equipment Corporation's VAX computers running under the VMS operating system. The first data base, ICMS, resides on a VAX 11/780 and has been implemented using VAX DBMS, a CODASYL based system. The second database, CSA, resides on a VAX 6460 and has been implemented using the ORACLE relational database management system (RDBMS).

Both databases are used for configuration management within the U.S. Navy. Different customer bases are supported by each database. ICMS tracks U.S. Navy ships and major systems (anti- air, weapons launch, etc.) located on the ships. CSA tracks U.S. Navy submarines and the major systems located on the submarines (anti-sub, sonar, etc.). Even though the major systems on ships and submarines have totally different functions, some of the equipment within the major systems are common to both ships and submarines.

2 PROBLEM STATEMENT

Even though the two databases are physically distinct, there a number of external actions which affect the data in both databases. Keeping the data consistent across these databases is a major problem. For example, the same computer is used within many major systems on ships and submarines. Thus, both CSA and ICMS maintain this information about this computer in their respective databases. If the U.S. Navy decides to stop buying parts for this computer from the existing supplier and start buying from an alternate supplier, this information needs to be updated in both databases. It should be noted that the two VAX's communicate via DECNET and that critical data must be updated in real-time up to 2 hours; all other data must be updated on a daily basis.

3 PROPOSED ALTERNATIVES

We propose two alteranatives.

(1) A global data manager (GDM) resides on one machine and all update transactions (to either of the systems) are routed through this central location. Any transaction that affects only one database is sent directly to the affected database. Since non-critical updates may be propagated on a daily basis, it is possible to send them directly to the affected database.

(2) Require each system to log all local transactions. Each system would have a local agent GDM that would regularly review each local database log and determine if a remote database transaction should be generated. The GDM would also report in the database log, the status of any transactions issued remotely. This alternative is the most viable for this environment. The GDM is primarily an initiator of transactions to keep the databases synchronized. The GDM also monitors the results of the initiated transactions and records the results in the originating database log. The local data managers (LDM) would be solely responsible for local transactions and local concurrency control.

The latter alternative appears to be more suitable for the current environment.

4 Sample Data

The ICMS database consists of records, data items and sets. An example of affected records sets and data items is given below.

```
SCHEMA NAME IS ICMS_SCHEMA

RECORD NAME IS SITE_DESCRIPTION
    WITHIN SITE_INFO
    ITEM SITE_IDENTIFICATION TYPE IS CHARACTER 15

RECORD NAME IS SYSTEM_DESCRIPTION
    WITHIN SYSTEM_INFO
    ITEM SYSTEM_NAME TYPE IS CHARACTER 26
    ITEM SYSTEM_AINAC TYPE IS CHARACTER 2
    ITEM SYSTEM_FSCM TYPE IS CHARACTER 5
    ITEM SYSTEM_PART_NO TYPE IS CHARACTER 26
    ITEM SYSTEM_STOCK_NO TYPE IS CHARACTER 26
    ITEM SYSTEM_CATEGORY TYPE IS CHARACTER 5

RECORD NAME IS SYSTEM_INSTALLATION_STATUS
    WITHIN INST_AREA
    ITEM SYSINST_STATUS TYPE IS CHARACTER 1
    ITEM SYSINST_SITE_ID TYPE IS CHARACTER 15
    ITEM SYSINST_SYSTEM_NAME TYPE IS CHARACTER 26
    ITEM SYSINST_SERIAL_NUMBER TYPE IS CHARACTER 15

SET NAME IS ALL_SITES
    OWNER IS SYSTEM
    MEMBER IS SITE_DESCRIPTION
     INSERTION IS AUTOMATIC RETENTION IS FIXED
     ORDER IS SORTED BY
         ASCENDING  SITE_IDENTIFICATION
         DUPLICATES ARE NOT ALLOWED

SET NAME IS SITE_SYSTEM_INST_STAT
    OWNER IS SITE_DESCRIPTION
    MEMBER IS SYSTEM_INSTALLATION_STATUS
     INSERTION IS AUTOMATIC RETENTION IS FIXED
```

```
     ORDER IS FIRST

SET NAME IS SYSTEM_SITE_INST_STAT
    OWNER IS SYSTEM_DESCRIPTION
    MEMBER IS SYSTEM_INSTALLATION_STATUS
     INSERTION IS AUTOMATIC RETENTION IS FIXED
     ORDER IS SORTED BY
         ASCENDING  SYSINST_SERIAL_LETTERS
                    SYSINST_SERIAL_NOS
         DUPLICATES ARE LAST

SET NAME IS ALL_SYSTEMS
    OWNER IS SYSTEM
    MEMBER IS SYSTEM_DESCRIPTION
     INSERTION IS AUTOMATIC RETENTION IS FIXED
     ORDER IS SORTED BY
         ASCENDING  SYSTEM_NAME
         DUPLICATES ARE LAST
```

The CSA database consists of tables and data items. Examples of tables and data items are given below.

```
TABLE: SITE

 Name                            Null?    Type
 ------------------------------- -------- ----
 SITE_IDENTIFIER                          CHAR(15)
 SQUADRON                                 CHAR(30)
 CONFIG_DATA_MANAGER                      CHAR(20)
 TYPE_COMMANDER                           CHAR(14)
 FLEET_CODE                               CHAR(1)
 STD_NAVY_DIST_LIST                       CHAR(8)
 LOCATION                                 CHAR(30)


TABLE: NOMEN_ITEM

 Name                            Null?    Type
 ------------------------------- -------- ----
```

```
CATEGORY_TYPE_CODE                NOT NULL CHAR(5)
PROGRAM_MANAGER                            CHAR(14)
TECHNICAL_MANAGER                          CHAR(14)
SYSTEM_INTEGRATION_AGENT                   CHAR(14)
DESIGN_AGENT                               CHAR(14)
ACQUISITION_ENG_AGENT                      CHAR(14)
TECHNICAL_DIRECTION_AGENT                  CHAR(14)
IN_SERVICE_ENG_AGENT_CODE                  CHAR(14)
IN_SERVICE_ENG_AGENT_PHONE_NUM             CHAR(12)
IN_SERVICE_ENG_AGENT                       CHAR(14)
SPCC_ITEM_MANAGER_PHONE_NUMBER             CHAR(12)
SPCC_ITEM_MANAGER                          CHAR(40)
DATE_UNDER_CONFIG_CONTROL                  DATE
DATE_OF_LAST_UPDATE                        DATE
TEST_AND_EVALUATION                        CHAR(14)
BRIEF_NAME                                 CHAR(50)
FULL_NAME                                  CHAR(70)
NOMEN                             NOT NULL CHAR(26)

TABLE: SITE_INSTALLATION

Name                              Null?    Type
--------------------------------- -------- ----
BELONGS_TO_SITE_ID                NOT NULL CHAR(14)
SERIAL_NBR                                 CHAR(15)
SERIAL_NUMBER                              CHAR(15)
SERVICE_APPLIC_CODE                        CHAR(10)
NOMEN                                      CHAR(26)
```

5 IMPLEMENTATION

The following implementation scheme is proposed to implement the second method.

A. Log all local transactions that relate to both databases. There will be one log per database and the log contains a timestamp of when the local transaction was committed, the node name, transactions (add, update, delete) and records/tables affected and data items affected. The key to all transactions is the system/unit nomenclature.

B. Modify each application program, that alters data, to log each transaction after it is committed. Sub-transactions must be entered into the log in the order they were executed on the local machine.

C. Generate list of system/equipment nomenclatures that are common between each database. The list would reside in a centralized location that would contain nomenclatures that are shared between each database. The Database Administrator (DBA) would maintain the nomenclature list.

D. Develop GDM to review the log files and search for transactions against common nomenclatures. After the GDM determines the transaction must be issued globally, it generates a remote transaction to be executed on the remote machine. GDM must know how to translate a transaction to be executed on another database.

E. The GDM must formulate the transaction in the appropriate data manipulation language so it can be executed on the affected database.

F. The GDM must send through DECNET a transaction package that will execute on the remote machine as a batch job. Set up account on each system for remote system to use to go into and submit batch jobs from.

G. The LDM will execute the transaction package as if it were generated locally.

H. When the transaction completes, the batch program will send a message back to the originating machine signifying that the transaction has completed. Once the message is received, the transaction is removed from the log. Generate a process to run on the other machine to update the log file.

I. Crash Recovery Procedures - As soon as a system recovers from a crash, a message will be sent to the other system to signal that processing can continue. The GDM will resend any transactions packages that it did not receive a completion on. If database commits and then crashes before sending a message, need to determine what to do.

6 RECOMMENDATIONS

The situation exists that two data bases that have multiple occurrences of the same data must learn to co-exist and keep each other informed of any changes to their duplicate data by keeping a log file, the internal database processing software is not affected. The application programs are responsible for logging the affects of the program.

AN INTEGRATED DECISION SUPPORT SYSTEM FOR TRAC: A PROPOSAL

Ravi Mukkamala
Department of Computer Science
Old Dominion University
Norfolk, Virginia.

ORGANIZATION

- INTRODUCTION
- EXISTING SYSTEM ARCHITECTURE
- PROPOSED SYSTEM
  - ARCHITECTURE
  - FEATURES
  - DEVELOPMENT PLAN
- PHASE I
  - OVERVIEW
  - BENEFITS
  - MAJOR TASKS
  - SCHEDULE
  - FEASIBILITY
  - EXTENSIBILITY AND MAINTAINABILITY
  - COST AND BENEFITS
- PHASE II - INCLUDE WORK PROGRAM AND MURS
- PHASE III - INTEGRATION OF DATA, MODELS, ETC.

INTRODUCTION

- Optimal allocation and usage of resources is a key to effective management.
- Resources of concern to TRAC are: Manpower (PSY), Money (Travel, contracts), Computing, Data, Models, etc.
- Management activities of TRAC include: Planning, Programming, Tasking, Monitoring, Updating, and Coordinating.
- Existing systems are insufficient; not completely automated; manpower intensive; potential for data inconsistency exists.
- Proposed system provides a means to integrate all project management activities of TRAC through the development of a sophisticated software and by utilizing the existing computing systems and network resources.

EXISTING SYSTEM ARCHITECTURE

- Independent systems for Study Program, Work Program, Manpower Utilization Reporting, Taskers, and Tracking Data Requests.
- Study Program:
  1. The system is developed using DBASE-3 on IBM-PC.
  2. This database only resides at RPD and is also accessed by RPD only.
  3. The system does not store the history of a study over its life time. It can only store information on three consecutive fiscal years (past, current, and next) at any given point.
  4. RPD mails the software on a floppy to TRADOC agencies. Users enter study request information using this software and send this information electronically or by US mail to RPD.
  5. RPD merges the requests and arrives at a consolidated list.

6. No history of changes to the study program is maintained. The original information is itself changed to reflect any new changes.
7. Status of the Studies in the program are printed once in a quarter and disseminated.

- Work Program:

1. No automatic linkage between the study program and work program exists at present.
2. Changes in the study program are not automatically reflected in the work program. Manual intervention is necessary to make these changes.

- Manpower Utilization Reporting:

1. An independent system, MURS, maintains manpower utilization information.
2. MURS is developed on DBASE-3 to run on a PC. It is being ported to run on Foxbase on Intel.
3. Each agency enters its manpower usage information and mails the floppy to TRAC RPD. RPD consolidates the information.
4. There is no automatic linkage between MURS and Work Program.

- Taskers:
  1. DBASE-3 based system is used to track status of taskers. There is no automatic link between the study program database and this system.
  2. Only RPD maintains it and uses it.

PROPOSED SYSTEM ARCHITECTURE

- A distributed database system - the TRAC agencies are the processing sites connected through a network.
- TRAC RPD at Fort Monroe is the principal site and contains additional software to maintain global consistency of data.
- The communication system consists of the Ethernets at the local sites, the IBM/SNA network, MILNET, and the networking through the Tracer.
- Each TRAC site contains a copy of the entire database.
- Non-TRAC sites will continue to interact with RPD by exchanging information non-interactively (e.g. Mail floppies).
- A site may contain either ORACLE software or DBASE-3 software.
- Each site functions autonomously for both data access and updates. In addition, a user at a site needs to access only the local copy of the database.

- The failure or unavailability of a site or communication channel will not affect the functionality of other available sites.
- The access rights to the data elements are clearly defined on a need-to-know basis. This will be implemented using the security features of the database software (ORACLE or DBASE-3) augmented with additional software developed by us.
- The distribution of data will be transparent to the TRAC users. That is, users need not be concerned about the data distribution.

PROPOSED SYSTEM FEATURES

1. The database is an integrated TRAC decision support system.
2. Updations made at a local site are automatically propagated to all relevant sites. To minimize network usage, updates will be propagated once or twice in a day.
3. In case a local site is down, it may be possible for users at that site to access database at other sites (or at least at the principal site).
4. Fast (local) access to data at any of the TRAC agencies.
5. A user friendly interface.
6. Multi-level security to the information in the database.
7. Fast and easy dissemination of information. E.g. Changes in Study Program.

8. Insure consistency within TRAC data.
9. Minimize duplication of efforts.
10. Ability to compare the actual versus planned utilization of resources.
11. On-line access to results (intermediate/final) from a study.

SUMMARY OF PLAN

- PHASE I: Since Study Program is the basis for a majority of TRAC activities, this will be taken up in this phase. In addition, this phase will solve the major problems related to distribution of data and communication between the sites. It will extend the existing Study Program information by adding Tasker scheduling information.
- PHASE II: This will extend the database to include Work Program and Manpower Utilization information.
- PHASE III: This extends the system to include data request tracking, model requests, monitoring of schedules, etc.

PHASE I - DEVELOPMENT

STUDY PROGRAM

- Guidance for AR 5-5 Study programs for next fiscal year.
- Requests for new or continuing studies for next fiscal year.
- Assign priorities.
- Authorize studies.
- Allocate resources.
- Tasker scheduling information.
- Update the program: termination, deletion, addition, updation, or completion.
- Past history to guide future programs.
- Handle arbitrary queries on the past, current, and next study programs.

PHASE I BENEFITS

- The latest guidelines are available on-line at the users' site.
- Possible to implement a complex priority scheme. This procedure can also be made available to the agencies.
- Since the study requests made so far are available at the RPD and other sites, it is possible to eliminate duplications at an early stage.
- By automatic linking of study authorization with resource allocation, speed and consistency can be achieved.
- Since agencies can update the status of a study locally, this information is more likely to be up-to-date.
- Since the status of a study is available to all agencies, other agencies can easily use this information for their planning.
- If reports, results etc. coming out of a study are made available at the agency's site, they can be easily retrieved by other agencies.
- Ability to access the data on previous programs increases the efficiency of future programs.
- Ability to handle arbitrary queries improves turnaround time of responses.
- It is possible not only to update a program, but also include comments indicating the reasons for the changes.

MAJOR TASKS IN PHASE I

1. **System Requirements:** An in-depth study of the Study Program and Taskers. It is also necessary to study how the decisions in this phase will influence development of other phases.
2. **ORACLE Database:** Study and experiment with the ORACLE software to determine its features, cost of these features, and its limitations.
3. **DBASE-3 Database:** Study and experiment with the DBASE-3 software to determine its features, cost of these features, and its limitations.
4. **Security:** Study and test the security features offered by ORACLE and DBASE-3. Check if these satisfy the TRAC's requirements. If not, determine ways to provide the additional security.

5. **Communications:** Study and experiment with the available communication facilities at TRAC. If the existing communication software cannot be directly used by the distributed database system, then some additional communication software may need to be developed to achieve the desired functionality: security, efficiency, and ease of use.

6. **Heterogeneity:** Since we are dealing with two different database software systems, we need to deal with the problems of heterogeneity. This requires the study of interoperability issues.

7. **Update Protocols:** Since the updates are made locally (at the sites) and sent to the primary site (RPD) at designated times, we need to develop a special monitor process that can accumulate the updates within a period and forward it to the primary site. Similarly, we need to develop a process at the primary site that can reliably propagate updates to all the TRAC sites. **Reliability** is the key. Issues such as the unavailability of a site, unavailability of a communication channel, and faulty communication channels need to be considered here.

8. **Consistency Checks:** In addition to the update protocols, it may be necessary to run some global consistency check protocols, probably run once in a week, to check the consistency of the global database.

9. **Communication Security:** Depending on TRAC's requirements, if the security offered by the existing communication systems for data transmission is insufficient, additional methods such as encoding or data compression may need to be developed and tested.

10. **Database Design:** Having studied the users' requirements, a database system needs to be designed. This requires the definition of the fields and their type, design of the relations, deciding the type of indexing, selection of additional secondary index tables, etc. To keep the data meaningful, we also need to arrive at some integrity constraints defined on the relations.

11. **Query and Form Design:** Write code (for ORACLE and DBASE-3) to execute the desired queries and print reports. This should be done at least for the existing reports and some known types of queries. However, a user may have to develop code for any special queries not covered in this phase.

12. **User Interface:** Depending on the facilities offered by ORACLE and DBASE-3 and depending on the sophistication of the TRAC's requirements, additional efforts may have to be expended to design a user interface to the decision support system.

13. **Implementation:** The system needs to be implemented on ORAACLE and DBASE-3. It also needs to be rigorously tested.

14. **Actual Data:** Once the testing phase is completed, the system is ready to be used. The actual data may now be placed into the system.

15. **User Training:** All TRAC users (or their representatives) need to be trained on system usage.

16. **Complaints/Comments:** We should also provide an automatic system to register any problems or improvements needed by the database system users. These should be looked into by the RPD personnel.

SCHEDULE FOR PHASE I

Activity	Time (in Weeks)	Period
System Requirements: Development	4	May'90
ORACLE Software: Study & experiment	3	May'90
DBASE-3 Software: Study & experiment	2	May'90
Database Security: Study & Develop	6	June-July'90
Communications: Study & Develop	5	June-July'90
Update Protocols	4	June-July'90
Communication Security	4	Aug'90
Database design	4	Aug'90
Consistency and Integrity	4	Sept'90
Query and Form Design	4	Sep'90
User Interface	6	Sep-Oct'90
Implement and Test	4	Oct'90
Incorporation of Actual data	2	Nov'90
Complaint/Comment Software	3	Nov-Dec'90
Documentation and User Training	4	Dec'90

FEASIBILITY

- ORACLE and DBASE-3 are two well established and proven database software systems. Since the proposed system is based on these, the chances of success are very high.
- A communication network system already exists between all TRAC sites. Even though it is only being used for Email, it should not be difficult to develop the required communication software on top of the existing system. The expertise of Mr. Hugh Dempsey, CIO, will be sought in developing this software.
- The staff at RPD are well versed with the existing systems. They are also clear on the inclusions to be made in the proposed system. This would make the system requirement development an easy task.

- The principal investigator, Ravi Mukkamala, is well versed with the issues and problems that arise in a distributed database system. Thus, protocol development for propagation of updates, data consistency, and handling site/link failures should not be difficult. Issues that require complete knowledge of ORACLE and DBASE-3 can be solved once these two systems are studied and experimented with.

- Implementing data security and communication security may be the most difficult tasks. Once again, we need to study the two database softwares and the communication systems to arrive at concrete code.

- Query and form design is not difficult. Both the ODU students that are committed (if paid) to this project are familiar with SQL. They have extensive experience in programming on DBASE-3. In addition, RPD staff currently handling the databases, are also very well versed with DBASE-3 programming.

EXTENSIBILITY AND MAINTAINABILITY

- Since the proposed system is based on a commercial database product, it should be easy to modify the database. The current RPD staff will be able to make changes related to: adding fields, creating additional relations, adding more queries, modifying the existing queries, etc.
- The problems in communications systems are more severe. They require the expertise of RPD staff such as Mr. Hugh Dempsey. If the communication system between the TRAC agencies is changed, then the communication software of this system also needs to be changed.
- Modifying the user-interface, once developed, may be more involved. The degree of difficulty will depend on its structure and the familiarity of the RPD staff with this software. Training one or two staff members at RPD may solve this problem.
- Extending the system from Phase-I to other phases should not be difficult due to the extensibility of relational database structures.

COST AND BENEFITS

- The ODU team will consist of the principal investigator and two graduate students. Each of the graduate students should be paid about $8,000 for the duration of the project. The principal investigator should be paid $15,000 as a summer salary and some release time during the Fall semester. Total: $31,000
- The ODU Research Foundation charges an overhead of 45%. In this case it would be $13,950. The overall cost up to Phase I is approximately $45,000.
- As a result of implementing the proposed system, TRAC can reduce duplication of efforts in maintaining different databases, achieve data consistency, and establish a strong base for an effective decision support system.

PHASE II - EXPANSION

INTEGRATION OF STUDY PROGRAM WITH WORK PROGRAM AND MURS.

PHASE III - FURTHER INTEGRATION

INTEGRATION OF DATA TRACKING, MODELS, AND SCENARIOS.

SUMMARY OF PLAN

- PHASE I: Since Study Program is the basis for a majority of TRAC activities, this will be taken up in this phase. In addition, this phase will solve the major problems related to distribution of data and communication between the sites. It will extend the existing Study Program information by adding Tasker scheduling information.
- PHASE II: This will extend the database to include Work Program and Manpower Utilization information.
- PHASE III: This extends the system to include data request tracking, model requests, monitoring of schedules, etc.