SimDS: A Simulation Environment for the Design of Distributed Database Systems

Alok R. Chaturvedi
Samir Gupta
Subhajyoti Bandyopadhyay
Purdue University

Abstract

Design of a distributed transaction processing system is a complex process. The paper describes the design and implementation of a general purpose scalable simulation environment (SimDS) for designing and evaluating the performance of distributed transaction processing systems. SimDS is a distributed simulation system where each data server is implemented as a separate process that communicates with each other through user datagram protocol. The paper describes the features of SimDS including various design policies that can be modeled by the system. It also discusses different design issues that one needs to consider in the implementation of a distributed simulation environment. The paper concludes with some test examples where SimDS was used to simulate different configurations of a real-time transaction processing system.

Keywords: information systems, database management, systems, distributed systems, physical design, simulation support systems, real time

ACM Categories: H.2, I.6

Introduction

Benefits accruing from a computer-based system critically depend on the business and computer management policies. These polices interact with each other and the overall benefit of the system depends on the complex interplay among these policies. As information systems' solutions differ from industry to industry and from application to application, there is a need for an environment that enables an IS designer or manager to evaluate different sets of policies and choose the best for the application at hand. This paper discusses the design and development of a simulation environment for evaluating management and operational policies for transaction processing systems. In particular, the paper focuses on a business environment that requires distributed real-time transaction processing (RDTPS).

A distributed system is one where data and computing power are located on several machines that are linked together by a network. The main advantages of a distributed system include increased availability of resources, increased reliability, and often increased speed of execution. In such systems, however, different computers coordinate activities to

complete the given task efficiently. Since each computer has a separate thread of execution, coordination of activities among machines is a complex task. The overall system benefit thus depends on the efficiency of coordination among machines. Nikolaou, Marazakis, and Georgiannakis (1997) have surveyed recent literature that discuss a number of different transaction routing mechanisms and their performance.

Further, in a transaction processing (TP) scenario, computers operate on large amounts of data that are typically resident on secondary storage devices, like disks. These systems are designed to guarantee data integrity under all conditions, irrespective of machine or network failures. Design of transaction processing systems focus on the efficiency of data retrievals and updates, increasing multiprocessing, and providing higher availability of data under different failure scenarios.

Generally, TP systems are required to complete each transaction that is initiated. In some cases, however, transactions are required to be completed prior to a given deadline (Haque and Wong, 1994). Such transactions are called hard real-time transactions. Examples of such transactions are computer initiated stock trading, threat analysis, defensive maneuvers, process control, etc. In these systems, a transaction may lose its value if it is not completed prior to its deadline. The performance measure of such systems is different from "normal" TP systems. These systems are geared towards optimizing the number of transactions completed prior to the deadlines.

It is clear from the above discussion that the design of a real-time, distributed TP system is extremely complex and requires a careful assessment of a large number of design and operational policies pertaining to each of the three areas — distributed processing, database issues, and real-time computing. The interactions among these policy choices are quite complex and difficult to predict. This paper describes a distributed simulation environment that has been developed to design and evaluate the performance of real-time distributed transaction processing systems. The remainder of the paper is organized as follows. The next section discusses the need for a simulation environment and discusses the related literature in this area. Section three describes the simulation environment and the implementation details. This section also delves into the reasons for opting for some of the mechanisms in the implementation of the simulation environment. Section four provides some examples and results of using this system for the design of a distributed TP system. The final section summarizes the important results and points to avenues for future research.

Need for a Distributed Simulation Environment

A survey of relevant literature indicates that researchers have attempted to study the performance of computer systems using one of the three approaches:

Analytical reasoning: In this methodology, researchers have attempted to represent the system as a mathematical model, generally as a queuing system.

Mechanistic: Another approach followed by many researchers is to propose and implement new mechanisms for one or more components of the overall system and determine the environmental conditions where the proposed mechanism would perform better than other policies.

Simulation: A third approach is where researchers have attempted to evaluate the performance of the database systems using simulation. Krishnamurthi, Basavatia, and Thallikar (1994) discuss one particular problem in such situations. Even though simulation models are validated and verified during the development process, a problem known as "deadlock" can still occur and go unnoticed in large, complex simulation models. Table 1 provides a comparative study of each solution paradigm and some of the relevant research in that area.

Need for Distributed Simulation

Multiple database sites can be simulated in a single physical program through establishment of virtual sites. For example, in an n site database system, one could create n sets of queues, with each set representing a data site. While such a design is simple to develop and maintain, a distributed simulation model is conceptually closer to the actual system and is more elegant from the programming point of view.

In a distributed simulation model, each data base site is simulated as a separate process that mayor may not reside on the same machine. Each site can be independently controlled, monitored, and altered. For example, the arrival rate of transactions and the data access times on site can be different from another.

Solution Paradigm	Advantages	Disadvantages	Research Effort
Analytical	Captures interations between subsystems	Most real life systems are too complex to be modeled mathematically	Chaturvedi et al. [CCR94] Lee & Liu Sheng [LL92]
	Can be used to predict the behavior of the system being modeled	Most sytems assume arrival to be independent and identically distributed	Gavish & Suh [GS92] Ram & Narasimhan [RN94] Wolfson [W87] Shu & Young [SY93] Yu et al. [YDL93] Ciciani et al. [CDY92] Rahm [R92] Bandyopadhyay et al. [BMS96] Bouras and Spirakis [BS96] Morrissey and Bandyopadhyay [MB95] Rahm [R93]
		Real-time distributed transaction processing systems do not exhibit this property	
Mechanistic	Focus on one or few components of system	Ignore interaction between different subsystems	Shin & Ramanathan [SR94]
	Develops new mechanisms to solve problems	Often does not consider business-related issues in design	Ramathirtham & Stankovich [RS94] Yu et al. [YWS94] Carey et al. [CJL] Dayal et al. [DB88] Sha et al. [SLJ] Davidson et al. [DLW] Segev & Park [SP89] Segev [S91] Schaad et al. [SSW95] Samaras et al. [SKC96] Rolia [R94]
Simulation	Easy to model complex systems	Critically dependent on accuracy of modeling	Abott & Garcia-Molina [AG88], [AG88A], [AG90], [AG92]
	Captures subsystems interactions well	Does not guarantee optimal solution	Thakore & Su [TS94] Hsiao & Dewitt [HD93]

Table 1. Comparative study of the various research paradigms.

The main reason for distributed simulation is that in cases where queues may became large and require large amounts of virtual memory, one can take advantage of virtual memory on several machines by running each process on a separate workstation. Distributed simulation also allows one to expand the research to study a very large database system comprised of several database sites by setting certain parameters in the software. Since the processes are not bound to a specific workstation, one can run an arbitrarily large simulation on a cluster of machines.

Secondly, Rajagopal and Comfort (1989) report a near perfect speedup using a distributed simulation model on a parallel machine. This feature can be exploited to study very large database systems.

However, there is a price for the greater flexibility and speedup. First, the system can be difficult and tricky to debug and validate. A distributed simulation system can have multiple threads of control, and there is a dearth of good debugging environments for concurrent processes. Secondly, the design depends quite heavily on interprocess communication. Communication of messages between processes that reside on different machines can be unreliable. In the final analysis, however, the conceptual clarity and scaleability of a distributed simulation system are well worth its programming complexity.

Simulation Environment: SimDS

A distributed simulation environment called SimDS has been developed for the design and evaluation of a distributed TP system. SimDS simulates a multisite disk-based database system that may or may not have real-time constraints. The database system is assumed to consist of several data servers or nodes. Each node has both computing and storage resources. The data are logically arranged as pages of memory. These pages may or may not be replicated on different nodes in the network. Transactions may arrive at any participating node. Each transaction has a release time and a deadline before which it needs to be completed (for transactions without real-time constraints this deadline is set to infinity).

Transactions may access any data item that exists in the database irrespective of the location of the data item. Each site possesses the complete data dictionary. When a transaction arrives, the node determines the entire read and write set for the transaction. The read-write set is then partitioned into sub-transactions and assigned to different sites for processing.

Once the sub-transactions have been processed independently at each assigned site, the results are returned to the processing site for computation. The processing site computes the results and then initiates the commit process. The transaction is said to be completed after the results have been computed and the commit process has taken place.

If one of the sites decides to abort a transaction, it informs the originating site, which then coordinates the abort process between all the sites. An aborting transaction has to release all locks that it had acquired. Thus the overhead of rolling back a transaction is the same as the time taken to release all locks.

Database Design Policies

There are many issues to be considered in the design of a real-time distributed transaction processing system (RDTPS). This section describes some of the policy parameters and the options that are considered in SimDS design. One of the more crucial issues in RDTPS deals with scheduling of transactions. For example, if there is more than one transaction ready to be executed by a computational resource, then which one would be scheduled first?

There are two scheduling issues that need consideration. The first deals with data consistency and serializability of transactions. If two transactions have a conflict between their read-write or write-write sets, then they cannot be concurrently executed and have to be serialized. SimDS simulates two widely accepted concurrency control policies:

Locking Based Policy: Under this option, each transaction requests and obtains access rights (called locks) prior to data access from the disk. These rights can be exclusive (for writing) or shared (for reading). The locks are released once the transaction is completed (Wolfson, 1987).

Optimistic Policy: Under this policy, transactions are permitted to continue execution without any kind of locking mechanism. However, after the transaction is complete and is ready to be committed, the system checks to ensure that it does not have a conflict with any of the uncommitted transactions. If there is

no conflict, then the transaction is committed, else some of the conflicting transactions are aborted to ensure serializability (Harista, Carey, and Livny, 1990).

The second set of scheduling issues is due to real-time constraints. Different transactions may have different opportunity costs or values and different deadlines. Some transactions may be so critical that they need to be executed at all costs. Others, however, may have a lower priority and can be aborted without unnecessary loss of value to the system. SimDS handles this issue through two different mechanisms. First, every transaction is assigned a priority value, and then the scheduler is designed to test different scheduling policies for transactions having varied priorities.

Priority allocation: Each transaction executed by the system is assigned priorities. These could remain unchanged throughout the execution of the program or may be recomputed at different stages of the life of the transaction. Priority is computed as a function of transaction attributes. Some of the policies modeled in SimDS include time of arrival (FIFO policy), deadline (Earliest Deadline Policy), slack in transaction execution (Least Slack Policy), and the economic estimate of the value of the transaction (Value Policy).

Scheduler Policies: These policies are identical for CPU and IO schedulers. Once priorities are assigned to transactions and the transaction enters either the CPU queue or the IO queue, the scheduler has to allocate the processor (CPU or IO processor) time for this transaction.
The scheduler may be required to make two decisions at this time:

To schedule the transaction or not: The scheduler estimates whether the transaction is likely to be completed prior to its deadline and chooses to schedule or abort the transaction on this basis. The following policies for scheduling are provided:

a. Ignore overload issue and schedule all transactions irrespective of their likelihood of completion (Schedule All Policy).

b. Abort all transactions in the system whose deadlines have elapsed (No Tardy Policy).

c. Abort all transactions which are unlikely to be completed before their deadlines (Feasible Policy). To achieve this, compute the estimated slack for the transaction, i.e., the difference between the deadline and the estimated processing time. If the slack is negative, abort the transaction and remove it from the queues.

To schedule a transaction if the resource is busy: If the processor is currently idle or is being used by a transaction having a higher priority, then the new transaction simply waits. If a higher valued transaction requests processor time, the scheduler resolves the contention depending on which of the following options is currently being exercised:

a. Non-preemptive Policy — Under this option, the current transaction is allowed to be completed before any new transaction is scheduled. (This is the default).

b. Preemptive Policy — Under this option, the current transaction is stopped and the new transaction is scheduled. The current transaction maintains its existing priority value.

c. Promoting Preemptive Policy — Under this option, the current transaction stops and lets the new transaction be scheduled. However, the current transaction assumes the priority value of the new transaction. This ensures that a low priority transaction does not repeatedly thrash in and out of the resource queue.

d. Conditional Non-Preemptive Policy — This policy allows the current transaction to be completed, if it estimates that the new transaction can "afford" to wait for this transaction to be completed. The slack value of the new transaction is computed, and if it is more than the estimated time of completion of the current transaction, the current transaction is allowed to execute.

SimDS provides the following policies for data fragmentation and replication:

No replication policy assumes that only one copy of any data item exists in the network. This is a viable policy option if the application is updated intensively and the network is relatively failure safe.

Full replication policy assumes that each site contains all data. This policy is viable for applications where response time is very crucial and where the underlying network is somewhat unreliable.

Partial replication policy refers to a policy mix between full and no replication. In this policy, different numbers of copies (or replicas) may exist for different data items.

All choices that can be simulated in SimDS are given in table 2.

SimDS Implementation

SimDS is comprised of two sets of servers — a name server called *Apex* and a set of database servers called *Dserv*. These servers are, by default, linked through a fully connected network (Figure 1). However, different network topologies can be simulated. *Apex* is the first server to be activated in any simulation run and exists on a well known location. This provides an anchor point for communication. *Apex* first creates a "receive port" where it listens for communication from a data server. *Apex* is also responsible for advancing the simulation clock, recording

Serialization	Replication	Commitment	Scheduling Overruns	Priority Assignment	Disk IO	Scheduling Concurrent Transactions
Central 2 Phase Lock	Full with Read One Write All (ROWA)	2 Phase	All	FIFO	FIFO	Wait
Primary site 2 Phase Lock	None	3 Phase	No Tardy	Earliest Deadline	Earliest Deadline	Wait and Promote
Central Optimistic	Partial with ROWA		Feasible Only	Least Slack	Least Slack	High Priority
Primary Site Optimistic						Conditional Restart

Table 2. Possible options in different design policies.

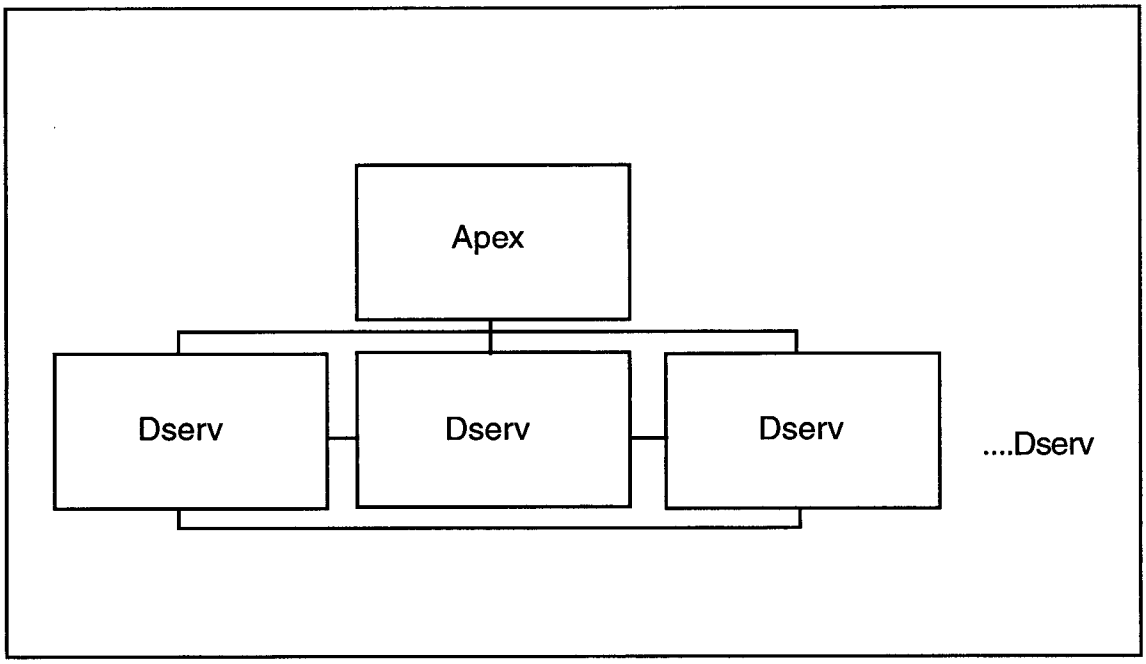

Figure 1. Topological design of the simulated system.

performance statistics for the entire system, and gracefully terminating each simulation run.

Next, a predefined number of data servers (*Dservs*) are activated. When each *Dserv* becomes active, it receives the global view of the system through handshaking. Figure 2 describes the sequence of messages for a two-data server system.

1. *Dserv 1* contacts *Apex* and communicates its receive port.
2. *Dserv 2* repeats the process.

3,4. *Apex* initializes all sites. This includes data/view materialization, communication of ports, location of data, global protocols like concurrency control policy, etc., and the clock initialization.

Simulation Design Choices

In this section we will describe some of the key components of the simulation environment.

Clock Management

The clock in SimDS is controlled by the name server — *Apex*. At the beginning of the simulation, *Apex* initializes the clock and broadcasts the new clock time to all data servers. When the new time is received, the data servers update their individual clocks and proceed to complete the activity that is scheduled for that time instant. Each data server computes the time of the next activity and sends this message to the name server. When *Apex* receives all "next activity times," it advances the clock to the minimum of these times and broadcasts the time.

Clock synchronization between different sites is maintained only for the purpose of simplicity. Lamport and Mellier-Smith (1985) and others have discussed the problems in clock synchronization in a distributed environment. In SimDS, therefore, a data server timestamps each incoming transaction with its own clock. All further processing within a data server is done using the local clock. Each data server also communicates with each other through an exchange of messages. Since each *Dserv* has an independent thread of control, it is difficult to ensure that all activity scheduled for a particular time is completed prior to the advancement of the clock. SimDS achieves the coordination by repeatedly transmitting the same clock time until there are no intersite messages between two successive "clock" broadcasts.

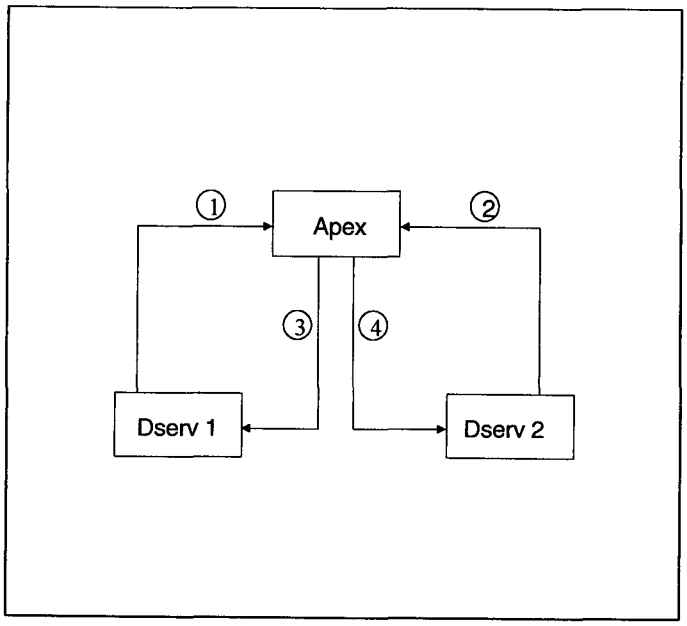

Figure 2. Handshaking between data servers and *Apex*.

A similar concept is used to abort and roll back transactions. Simulation of transaction abortion is a difficult problem since it is hard to predict whether a transaction will abort at one site or at multiple sites, and at what time. In SimDS, an aborted transaction is not erased from the resource queues of each site immediately. When a transaction is aborted, its state is immediately changed to ABORT. The transaction, however, remains in the system until all intersite communication that occurs at that clock instant has been completed.

Transmission Control Protocol

SimDS has been developed in 'C' on a UNIX platform. It uses the user datagram protocol (UDP) for interprocess communication. The alternate choice was transmission control protocol (TCP). UDP is a connectionless communication protocol as against TCP which first establishes connection between the two parties prior to message transfer. While TCP provides a greater degree of reliability compared to UDP, UDP was found to be adequate for this application. UDP was observed to be quite reliable when the host machines were connected on a small LAN. UDP also provides a greater throughput than TCP, enabling the simulation to run faster.

Service Queues

Each database site in the simulation maintains a transaction queue and three resource queues (one for each the CPU, disk IO, and network messages). The transaction queue maintains a list of transactions and sub-transactions that are alive at a particular site. Each transaction is identified by the site of origin and a transaction serial number assigned by the originating site. In the transaction queue, multiple entries of the same transaction can exist simultaneously. For example, the transaction queue on the originating site may contain an entry for the main transaction, one for the part of the transaction that pertains to the data resident on that site, and another for a request for data lock (if the site is also a lock site).

The resource queues maintain a list of transactions or sub-transactions that are either using the particular resource or are waiting to use the resource. The current version of SimDS assumes that there is only one CPU and one disk at each site. The software could easily be extended to study multi-process and multi-disk data servers.

The network queue is used to implement network delays. It can also be extended to simulate network failures — network partitioning and node failure. When a site wishes to transmit a message to another site, it enqueues the message in the message queue with some estimated delay. After the delay interval, the message is transmitted to the required site.

Transaction splitting and allocation

In a RDTPS, a transaction may need to be processed at multiple sites. SimDS simulates this by splitting a transaction and assigning it to multiple data sites. Two different policy considerations are implemented in SimDS:

1. *Number of sites to allocate the transaction:* This depends on the read-write policy simulated by the system at the time. In the first policy (Read One Write All), each transaction is read from only one copy of the data while the update is performed on all sites that contain the data item. In a quorum-based policy, multiple sites are read and written.
2. *Allocation of transaction:* The actual assignment of transactions to the different sites is done on the basis of the assignment policy simulated by the system. The assignment can be done either randomly or based on the transaction and environmental parameters. Handles are provided to incorporate intelligent allocation of transactions.

Atomicity of Broadcasts

One of the requirements in a distributed system is to maintain the atomicity of broadcasts, i.e., any causal relationship between two messages need to be maintained at the receiving end. In other words, if site 1 receives message A and uses this message to transmit message B, then another site 2 on the network must logically receive message A prior to message B.

The network delay estimation process at each node is responsible for maintaining the atomicity of broadcasts. When a new message is generated, the delay estimation process scans the outstanding message queue for the last message (M) that pertains to the same transaction and has the same destination. The delay estimate of the new transaction is set to be greater than the delay of the message M.

Deadline Estimation

SimDS estimates the deadline for each transaction on the basis of the parameters specified for each site. This estimate is generated at the time the transaction is created and is constant throughout the life of the transaction. SimDS generates deadlines through a user-defined uniform distribution.

Estimation of Transaction Completion Time

The time taken for each step in the processing of transactions is determined externally through a parameter file. The estimated time of processing a transaction is the sum total of all sub-steps plus expected waiting time in one or more resource queues. (The default value of the expected waiting time is set to zero).

Priority Allocation

SimDS computes priority of each transaction at the time a transaction is created. Abott and Garcia-Molina (1988) have indicated that a static priority assignment works at least as well as a dynamic policy. Thus once a priority value is computed, it remains unaltered throughout the life of the transaction.

SimDS generates priorities from a multimodal distribution. The distribution is a composite of many uniform distributions, each having a usage probability associated with it. The system first chooses a distribution from this set of distributions using a uniformly distributed random number (0,1). Next, a priority estimate is computed based on the distribution chosen in the earlier step. This enables the user to simulate a mixture of different classes of transactions, such as emergency, important, and routine.

Scheduling of Transactions in CPU and IO Queues

When a transaction needs to use a resource, it is enqueued in the resource queue. Whenever the particular resource is free, its scheduler scans the queue to pick the transaction with the highest priority. The transaction is then verified to check if it can be scheduled on the basis of scheduling policies. If needed, the originating site of the transaction is informed to coordinate a clean abort. The scheduling policies for IO and CPU queues are independent of each other. Users can specify different prioritizing schemes for "read-only transactions" and "update transactions" in the IO queue.

Locking Policy

In a central site system, the system assigns a lock site at the start of the simulation. In a primary site system, one or more sites could be used to perform locking on different subsets of data. When a transaction requires a lock, it obtains it by requesting locks from all sites that are responsible for maintaining locking for the data associated with the transaction. Locking granularity is held constant at a page of memory (Abott and Garcia-Molina, 1988).

The lock site maintains the number of shared and exclusive locks it has granted. This is used to compute the number of pages locked by transactions. When a new request is received, the lock mechanism either grants the lock or refuses it using the fraction of database locked by existing transactions. When a transaction terminates, the lock mechanism reduces the number of pages locked.

Simulation Parameters

This section describes the different parameters used by SimDS.

Arrival Rates: A user can specify the transaction arrival rate at each site. The inter-arrival time is set to an exponential distribution with the mean as the reciprocal of the arrival rate.

Transaction Sizes: The default for granularity of database size is a page of memory. (The system could be set to change the granularity to a tuple of a data table). For example, each transaction will either read or write (or both) a number of pages of memory that were determined by a normal distribution with a mean of 12 pages and a variance of four pages. (The smallest data size affected by each transaction was off-course zero).

Volatility: Volatility is the measure of the fraction of transactions that updates the database as a percentage of the total number of incoming transactions.

Number of Data Sites: SimDS is a scalable system and can simulate a very large system. The total number of data sites that can be simulated is restricted by the operating system and network considerations. Each process maintains an open socket for each of the other processes. Since the operating system restricts the number of sockets that can be opened simultaneously, it governs the size of the system that can be simulated. The performance of the system is seen to grow linearly with the number of data sites and does not restrict scaling.

Concurrency Control Protocols: SimDS is capable of simulating two of the most widely studied concurrency control protocols — locking based and optimistic. Furthermore, SimDS simulates centralized or distributed concurrency control. For example, one can designate data site "one" as the only lock site. Thus all locking tables would reside at site "one" and all sites request lock from this site.

Locking Mechanism: Each transaction determines its read-write set at the very outset (at the completion of the precompute stage) and requests and releases all locks at one time. Such an assumption allows one to ignore the simulation of deadlock detection and prevention/avoidance and simplifies the state transition of the system to a large extent.

SimDS currently simulates a uniformly distributed access pattern, i.e., transactions can access any of the data items in a database. A lock is granted to a transaction if there is no overlap between its read sets and the write sets of all transactions currently holding locks or between its write sets and the read sets of the existing transactions. If a lock is denied, SimDS can currently restart the transaction, or it blocks it for a fixed period of time.

Scheduling Protocols: In a real-time transaction processing environment, scheduling of different transactions is crucial to performance of the system. SimDS incorporates four different scheduling strategies: FIFO, earliest deadline first, least slack first, and maximum "benefit" first. Related to these algorithms is the strategy for preemption of current transaction. SimDS is designed to simulate any of the three given strategies: Wait, Preempt, and Preempt and Promote.

IO Protocols: Scheduling strategies affect the allocation of the CPU to each transaction. A similar set of strategies allow scheduling of transaction that are waiting to complete disk I/O. Additionally, the user can opt for different strategies for read operations as against write operations.

Deadline Estimation: The estimation of deadlines in SimDS is set to a composite distribution comprising of one or more uniform distributions.

Observation Parameters

The server *Apex* maintains all results observed in the system. When a transaction is committed or aborted, the originating site informs *Apex* to update the statistics. At the end of the simulation run, *Apex* sends a terminating message to the data sites. At this point each site transmits a message to *Apex*, informing it of the statistics pertaining to the resource queues.

Included among the different statistics computed by the system are:

- Number of transactions completed prior to the deadline.
- Throughput — the number of transactions completed prior to the deadline divided by the simulation clock elapsed.
- Total delay observed by all transactions.
- Mean queue lengths for each site and each resource.
- Mean time spent in the resource queue by each transaction at each site and for each resource.

Experiment Design

This sections demonstrates how RDTPS experiments can be run using SimDS. Experiments conducted in this set simulates a total of 600 transactions. The first 100 transactions were ignored to allow the system to reach a steady state. The stability of the system was checked by examining the length of resource queues periodically during the simulation run. Arrival rates beyond which the system became unstable were excluded from the study.

Bias originating from the generation of random number sequence was eliminated by conducting each experiment several times with different "seed" values. Different system parameters like average throughput, number of transactions completed prior to the deadline, etc., were measured. The results are the average value determined by each set of experiments.

Examples and Discussion

This section presents some sample results obtained in the study. SimDS was used to simulate a two-data site system where the data were partly replicated. Single site locking was used as the serializability mechanism. Experiments were conducted to study the impact of scheduling policies, priority assignment, and concurrency conflict resolution policies on different system parameters.

Impact of Load on the System

In this set of experiments, the impact of increasing the arrival rates was observed on the throughput of the system and the number of transactions completed prior to the deadline. Figure 3 presents the results obtained.

Clearly, the most striking feature of the above chart is the difference in behavior of the system when all transactions are scheduled against when tardy transactions are aborted. When the system decides to schedule all transactions, the throughput initially increases with arrival rate but it drops rapidly at very high loads. This behavior was observed under all combinations of priority assignments and preemption policies. The behavior of the system is markedly different when the scheduling policy changes from "All" to "Feasible" or "Alive Only." In these cases the throughput continues to rise with increase in arrival rates. This is again an expected result since the scheduling policies prevent overloading of the system.

Another interesting statistics is the number of transactions completed within stipulated time as a fraction of the number of transactions that enter the system. Figure 4 presents that result.

The number of transactions completed prior to deadlines shows a marked decline when all transactions are scheduled. When the system tries to schedule only feasible transactions, the number of transactions completed within stipulated time is fairly stable. When the system schedules all transactions that have not reached the deadline, the number of "correct" transactions shows a decline, although a small one. Also, it appears that the fraction of correct transactions reaches a stable value at high system loads. This is quite reasonable since the scheduling policies prevent system overload.

Impact of Scheduling Policies

In this set of experiments, the scheduling policies and the arrival rates were altered, and the impact on the system performance under different priority assignments and abort policies was observed. The results obtained in all categories were identical to the ones represented in Figures 3 and 4.

Apart from the system behavior at high system loads, it is also interesting to study the performance of the scheduling policies in a lightly loaded system. At low arrival rates, scheduling all transactions yields the highest throughput and the highest fraction of "correct" transactions. Clearly, deliberately aborting a transaction since it is not expected to be completed by its deadline is an overkill and deteriorates the system performance. The "No Tardy" scheduling policy on the other hand yields throughput rates close to the policy of scheduling all transactions. Thus under these conditions, the "No Tardy" policy yields good results at all system loads.

Impact of Priority Assignment

In this set of experiments, system behavior was studied under the influence of different priority assignment policies. The transactions in the CPU queue were prioritized according to the governing priority assignment policy while the IO transactions were scheduled on a FIFO basis. Figure 5 illustrates the results obtained.

The above results were obtained for the case where the system would schedule all transactions. It is clear that the priority assignment policy did not have any significant effect on the behavior of the system. A similar result was obtained for all other scheduling policies. To get a deeper understanding of this result, some snapshots of the process queues were studied at random points during the simulation. It was observed that the ordering of transactions in each queue was almost independent of the priority assignment policy.

A more clear understanding of the impact of priority assignment policies is obtained from examining the CPU queue of both servers. The average number of transactions waiting for the CPU was observed to be less than one, even for fairly high loads. CPU priority assignment policies will have an impact on the throughput if two or more transactions have to contend for the CPU at a given time. Clearly, there is very little contention for the CPU resource, even at very high loads.

Priority Assignment in the IO Queue

In this set of experiments, we altered the priority assignment of transactions and used these assignments to schedule transactions in the IO queue. Figure 6 yields the results obtained from this set of experiments.

The priority assignment policies for IO queues have

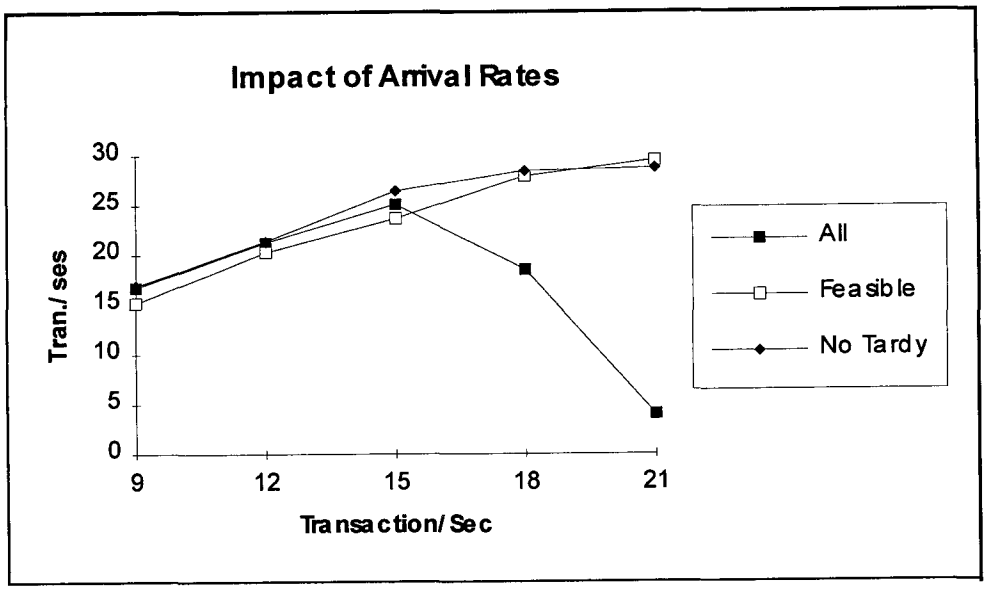

Figure 3. Impact of system load on throughput of the system.

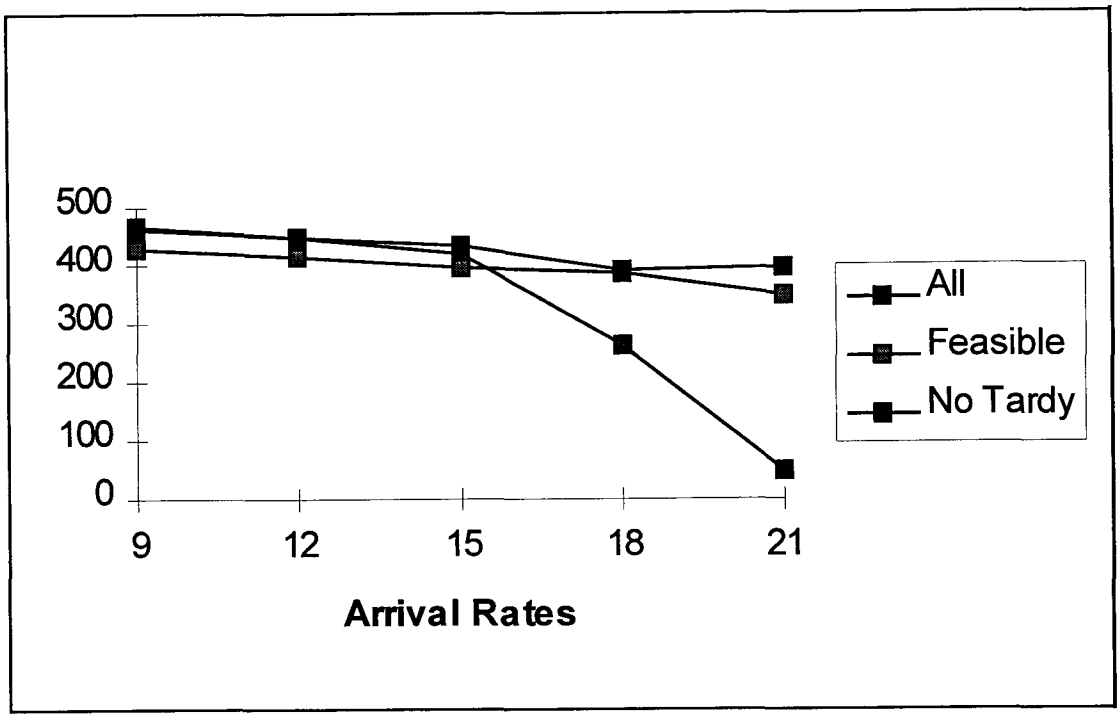

Figure 4. Impact of system load on number of transactions completed within deadline.

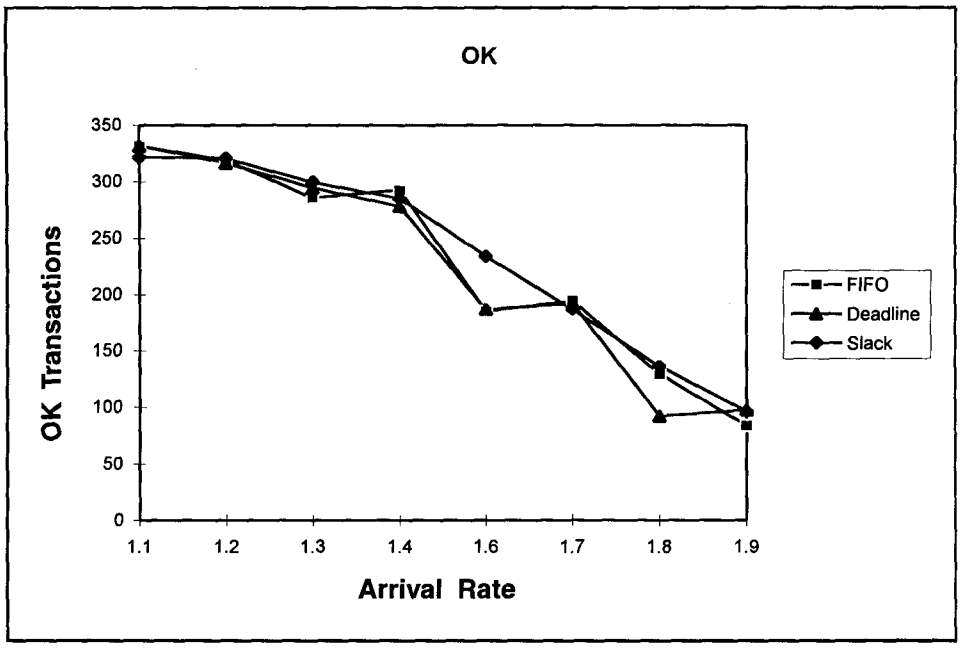

Figure 5. Impact of CPU priority assignment on number of completed transactions.

a significant impact on the number of transactions completed prior to the deadline. The difference in the performance of the system under the three policies increases with an increase in the load on the system. At low loads, all policies yield similar throughputs. The behavior is easily explained by examining the number of transactions waiting in the IO queue. Figure 7 illustrates the change in the waiting time in the IO queue.

At low loads, the average number of transactions waiting in the IO queue are less than one. Thus the priority assignment policies will not impact the performance of the system. At high loads however, the mean length of the queues increase significantly and therefore the scheduling of transactions is crucial.

Interestingly, the policy of prioritizing the transactions based on the "least slack" policy performs better than the "FIFO" policy while the "earliest deadline" policy performs worse than "FIFO." Since slack is defined as the difference between deadline and the estimate of processing time, the exact determination of processing estimates is crucial to the system performance. However, processing time estimation has certain costs associated with it. The authors propose to explore the trade-offs involved in processing time estimation and the benefits that accrue from such estimates in future research.

Impact of Lock Conflict Resolution Policy

In this set of experiments, we studied the impact of lock conflict resolution on the behavior of the system under different scheduling polices and varying loads. Figure 8 illustrates the results obtained.

Under very light load situations, there was some difference in the behavior of the system in different situations, but in heavier loads the scheduling policy governs the behavior.

Upon deeper observation, it became clear that the performance of the system is quite independent of the conflict resolution policy used since the transaction can be rolled back only at one point in its life — just after preprocessing. Since the CPU is not the bottleneck in the system, rolling back a transaction is not critical to system performance. There are normally two orders of magnitude difference between the rollback time and the blocking time (since block time is equal to expected transaction processing time including IO time). Therefore, the main determinant of the system performance is the waiting time of transactions in the IO queue.

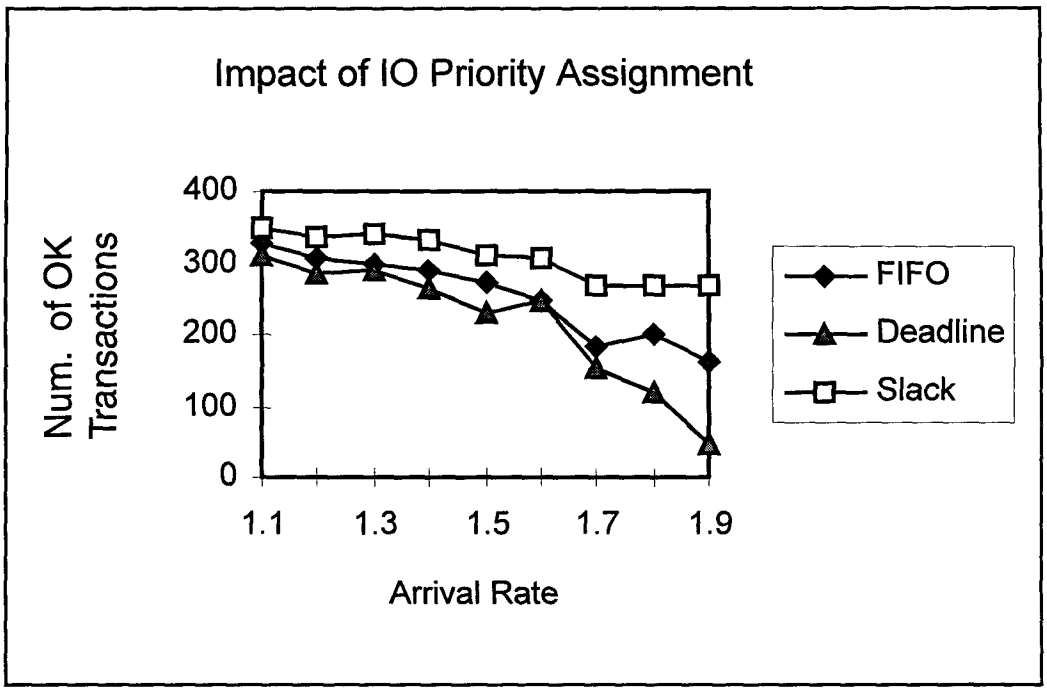

Figure 6. Impact of IO priority assignment.

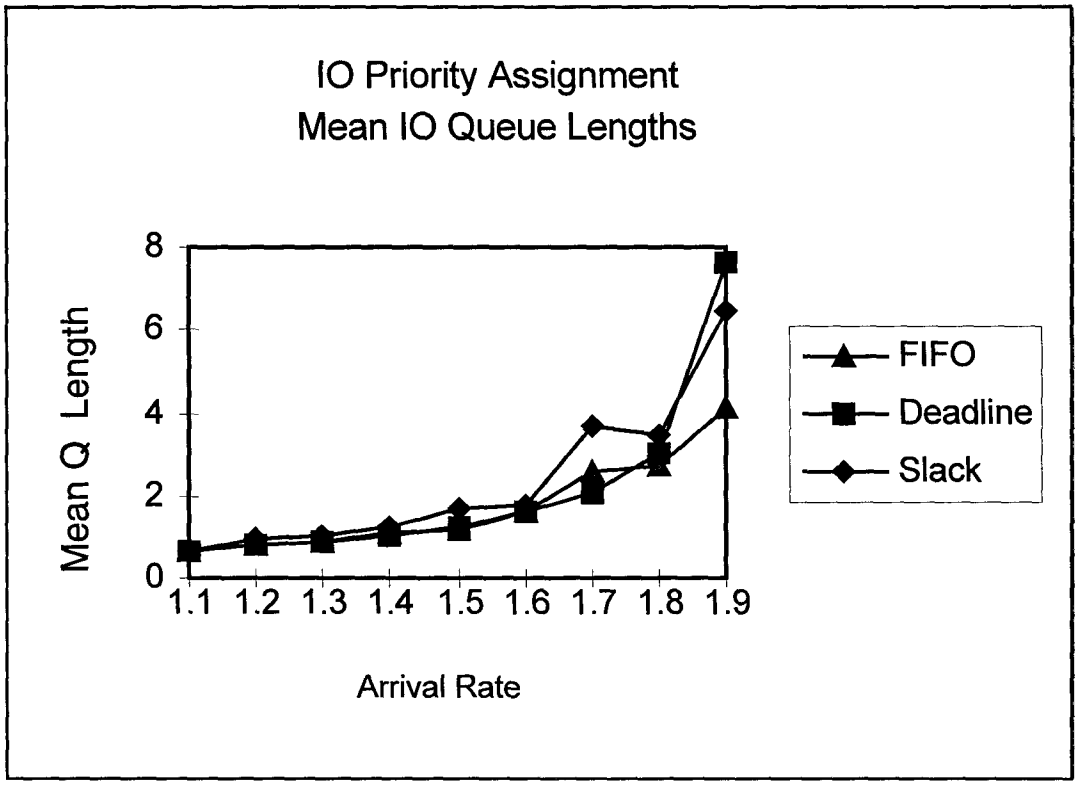

Figure 7. Impact of IO priority assignment on resource queues.

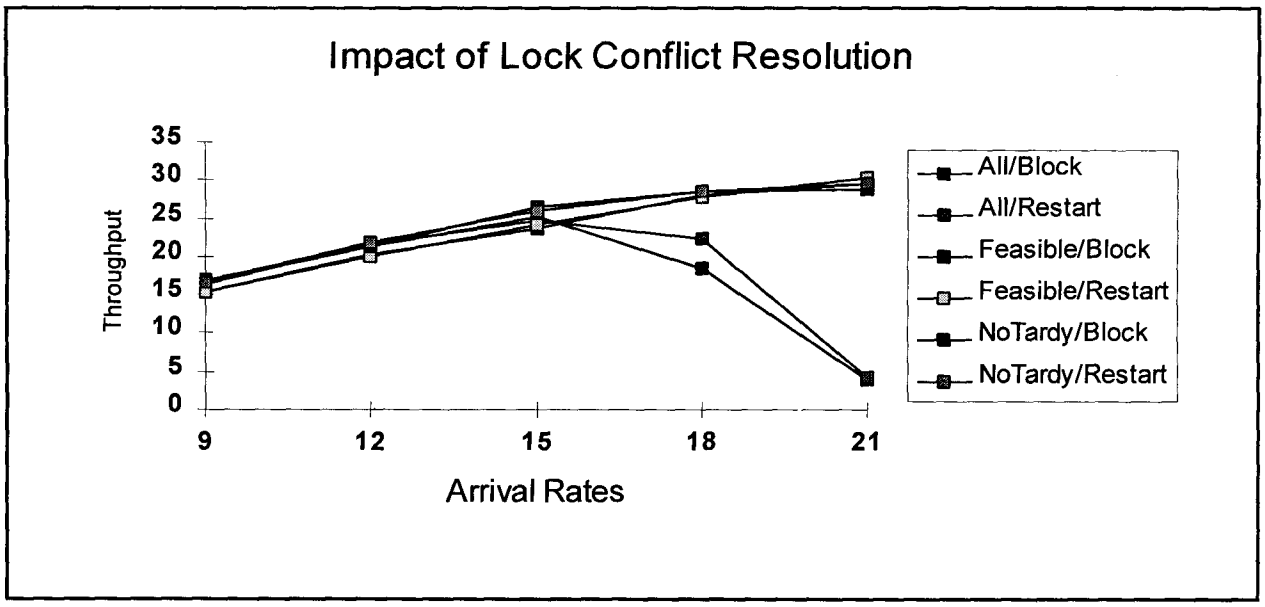

Figure 8. Impact of conflict resolution policy.

Conclusion and Future Research

While the above study was intended to demonstrate the capabilities of SimDS, it provided the following significant results.

The SimDS environment is quite conducive to the study of the performance of distributed TP systems under varying conditions and policies. In its present form, it simulates a variety of CPU and IO scheduling policies, transaction priority assignments, centralized and distributed serialization, and different replication and read-write policies for real-time distributed systems connected by different network topologies. Sample experiments were conducted to study the impact of CPU and IO scheduling, transaction priori-tization, and concurrency conflict-resolution policies for different systems loads in a two-site system. Studies suggest that the throughput of the system is almost solely governed by the waiting time of transactions in the IO queue.

SimDS is an ideal tool in the design and evaluation of a distributed TP system under different environmental and operational conditions. Some of the studies that are currently being done using SimDS include impact of network delays and scalability of RDTPS performance, the cost and benefit of providing intelligence in resource allocation within a data site, and the allocation of transaction to data sites; the "best" set of design and operational policies under different applications and environmental conditions. These include concurrency control and scheduling policies, data and transaction allocation, and centralized and distributed control of the RDTPS.

Bibliography

Abott, R.K., and Garcia-Molina, H. (1988). "Scheduling Real-time Transactions," *ACM SIGMOD Rec.*, pp. 71-81.

Abott, R.K., and Garcia-Molina, H. (1988) "Scheduling Real-time Transactions: A Performance Evaluation," *Proceedings of the 14th VLDB Conference*, pp. 1-12.

Abott, R.K., and Garcia-Molina, H. (1990). "Scheduling I/O Transactions with Deadlines: A performance Evaluation," *IEEE Real-Time System Symposium*, pp. 113-124.

Abott, R.K., and Garcia-Molina, H. (1992). "Scheduling Real-time Transactions: A Performance Evaluation," *ACM Transactions on Database Systems*, Vol. 17, No. 3, pp. 513-560.

Agrawal, R., Carey M.J., and Livny M. (1987). "Concurrency Control Performance Modeling: Alternatives and Implications," *ACM Transactions on Database Systems*, Vol. 12, No. 4, pp. 609-654.

Bandyopadhyay, S, Morrissey, J., and Sengupta, A. (1996). "Query Optimization Strategy for Distributed Databases on All-optical Networks," *Canadian Conference on Electrical and Computer Engineering*, Vol. 1, IEEE, pp. 245-248.

Bouras, C.J., and Spirakis, P.G. (1996). "Performance

Modeling of Distributed Timestamp Ordering: Perfect and Imperfect Clocks," *Performance Evaluation*, Vol. 25, No. 2, pp. 105-130.

Carey, M., Jauhari, R., and Livny, M. (1989). "Priority in DBMS Resource Scheduling," *Proceedings of the 15th VLDB Conference*, pp. 397-410.

Chaturvedi, A.R., Choubey, A.K, and Roan, J. (1994). "Scheduling the Allocation of Data Fragments in a Distributed Database Environment: A Machine Learning Approach," *IEEE Transactions on Engineering Management*, Vol. 41, No. 2, pp. 194-207.

Chen, Graham. (1985). "Distributed Transaction Processing Standards and Their Applications," *Computer Standards & Interfaces*, Vol. 17, No. 4, pp. 363-373.

Ciciani, Bruno, Dias, Daniel M., and Yu, Philip S. (1992). "Analysis of Concurrency-coherency Control Protocols for Distributed Transaction Processing Systems with Regional Locality," *IEEE Transactions on Software Engineering*, Vol. 18, No. 10, pp. 899-914.

Davidson, S., Lee, I., and Wolfe, V. "A Protocol for Timed Atomic Commitment," *IEEE Conference on Distributed Computing Systems*, pp. 199-206.

Dayal, U., Blaustein, B., et al. (1988). "The HiPAC Project: Combining Active Databases and Timing Constraints," *ACM SIGMOD Rec.*, pp. 51-70.

Gavish, B., and Suh, M.W. (1992). "Configuration of Fully Replicated Distributed Database System over Wide Area Networks," *Annals of Operation Research*, Vol. 36, pp. 167-192.

Haque, Waqar, and Wong, Johnny (1994). "Distributed Real-time Nested Transactions," *Journal of Systems & Software*, Vol. 27, No. 2, pp. 85-95.

Harista, J., Carey, M., and Livny, M. (1990). "On Being Optimistic about Real-time Constraints," *Proceedings of ACM Symposium on Principles of Database Systems*, pp. 331-343.

Harista, J., Carey, M., and Livny, M. (1990). "Dynamic Real-time Optimistic Concurrency Control," *IEEE Real-time Systems Symposium*, pp. 94-103.

Hsiao H., and Dewitt, D. (1993). "Performance Study of 3 High Availability Data Replication Strategies," *Distributed and Parallel Databases*, Vol. 1, No. 1, pp. 53-80.

Huang, J., et al. (1990). "On Using Priority Inheritance in Real-time Databases," *COINS TR 90-121*, Department of Computer Science, University of Massachusetts.

Iyer, B., Yu, P., and Lee, Y. (1986). "Analysis of Recovery Protocols in Distributed On Line Transaction Processing Systems," *IEEE Real-time Systems Symposium*, (December), pp. 226-233.

Krishnamurthi, Murali, Basavatia, Amar, and Thallikar, Sanjeev (1994). "Deadlock Detection and Resolution in Simulation Models," *Winter Simulation Conference Proceedings*, IEEE, pp. 708-715.

Lamport, L., and Melliar-Smith, P.M. (1985). "Synchronization of Clocks in the Presence of Faults," *Journal of ACM*, Vol. 32, No. 1, pp. 52-78.

Lee, H., and Liu Sheng, O.R. (1992). "A Multiple Criteria Model for the Allocation of Data Files in a Distributed Information System," *Computers & Operation Research*, Vol. 19, No. 1, pp. 21-33.

Morrissey, J. M., and Bandyopadhyay, S. (1995). "Computer Communication Technology and Its Effects on Distributed Query Optimization Strategies," *Canadian Conference on Electrical and Computer Engineering*, Vol. 1, IEEE, pp. 598-601.

Nikolaou, C. N., Marazakis, M., and Georgiannakis, G. (1997). "Transaction Routing for Distributed OLTP Systems: Survey and Recent Results," *Information Sciences*, Vol. 97, No. 1-2, pp. 45-82.

Rahm, Erhard (1992). "Framework for Workload Allocation in Distributed Transaction Processing Systems," *Journal of Systems & Software*, Vol. 18, No. 2, pp. 171-190.

Rahm, Erhard (1993). "Evaluation of Closely Coupled Systems for High Performance Database Processing", *13th International Conference on Distributed Computing Systems Proceedings — International Conference on Distributed Computing Systems*, IEEE, pp. 301-310.

Rajagopal, R., and Comfort, J.C. (1989). "Contrasting Distributed Simulation with Parallel Replication: A Case Study of a Queuing on a Network of Transputers," *Proceedings of 1989 Winter Simulation Conference*, pp. 746-755.

Ram, S., and Chastain, C.L. (1989). "Architecture of Distributed Data Base Systems," *Journal of Systems & Software*, Vol. 10, pp. 77-95.

Ram, S., and Narasimhan, S. (1994). "Database Allocation in a Distributed Environment: Incorporating a Concurrency Control Mechanism and Queuing Costs," *Management Science*, Vol. 40, No. 8, pp. 969-983.

Ramathirtham, K., and Stankovic, J.A. (1994). "Scheduling Algorithms and Operating System Support for Real-time Systems," *Proceedings of IEEE*, Vol. 82, No. 1, pp. 55-66.

Rolia, J. A. (1994). "Distributed Application Performance, Metrics and Management," *IFIP Transactions C: Communication Systems*, No. C-20, pp. 235-246.

Samaras, George, Kshemkalyani, Ajay D., and Citron, Andrew (1996). "Context Management and Its Applications to Distributed Transactions," *Proceedings — International Conference on Distributed*

Computing Systems, IEEE, pp. 683-691.
Schaad, Werner, Schek, Hans-J., and Weikum, G. (1995). "Implementation and Performance of Multi-level Transaction Management in a Multidatabase Environment," *Proceedings of the International Workshop on Research Issues in Data Engineering — Distributed Object Management — RIDE-DOM,* IEEE, pp. 108-115.
Segev, A. (1991). "Strategies for Distributed Query Processing," *Information Sciences,* Vol. 54, Nos. 1-2, pp. 67-88.
Segev, A., and Park, J. (1989). "Updating Distributed Materialized Views," *IEEE Transactions of Knowledge and Data Engineering,* Vol. 1, No. 2, pp. 173-184.
Sha, L., Lehoczky, J., and Jensen, E. "Modular Concurrency Control and Failure Recovery," *IEEE Transactions on Computers,* Vol. 37, pp. 146-159.
Shin, K., and Ramanathan, P. (1994). "Real-time Computing: A New Discipline of Computer Science and Engineering," *Proceedings of IEEE,* Vol. 82, No. 1, pp. 6-23.
Shu, L., and Young, M. (1993). "Real-time Concurrency Control with Analytic Worst Case Latency Guarantees," *Proceedings of 10th IEEE Workshop on Real-Time Operating Systems and Software,* New York, pp. 66-73.
Thakore, A., and Su, S. (1994). "Performance Analysis of Parallel Object-oriented Query Processing Algorithm," *Distributed and Parallel Databases,* Vol. 2, No. 1, pp. 59-100.
Wolfson, O. (1987). "The Overhead of Locking (and Commit) Protocols in Distributed Databases," *ACM Transactions on Database Systems,* Vol. 12, No. 3, pp. 453-471.
Yu, Philip S., Dias, Daniel M., and Lavenberg, Stephen S. (1993). "On the Analytic Modeling of Database Concurrency Control," *Journal of the ACM,* Vol. 40, No. 4, pp. 831-872.
Yu, P.S., Wu K., Lin K., and Son S.H. (1994). "On Real-time Databases: Concurrency Control and Scheduling," *Proceedings of IEEE,* Vol. 82, No. 1, pp. 140-157.

About the Authors

Alok R. Chaturvedi is an associate professor of management information systems and the Director of Synthetic Environments for the Analysis and Simulation Laboratory at the Krannert Graduate School of Management, Purdue University. He received his Ph.D. in management information systems and computer science from University of Wisconsin-Milwaukee. He has published extensively in major journals and conference proceedings. He also has extensive consulting experience with companies such as AT&T, IBM, Ameritech, Bell Atlantic, Abbott Labs, Ernst & Young, the U.S. Congress, and the Department of Defense.
E-mail: alok@mgmt.purdue.edu.

Subhajyoti Bandyopadhyay is a second year Ph.D. student in MIS at the Krannert Graduate School of Management, Purdue University. He received his B.Tech. degree in electrical engineering in 1991 from the Indian Institute of Technology, Kanpur, and in management in 1993 from the Indian Institute of Management, Bangalore. Prior to joining the Ph.D. program at Purdue University, he worked in various management positions at IBM, India, from 1993 to 1997. His interests include the economics of electronic commerce, pricing of informational and digitized goods, synthetic agents, and economies. He is currently working on the emerging business of digitized music and its effect on the various parties involved.
E-mail: shubho@purdue.edu

Samir Gupta is a visiting assistant professor at Purdue University, Indiana. His current research interest lies in designing distributed transaction processing systems that may or may not have real time constraints. He has also worked in the area of data visualization in a financial environment.
E-mail: gupta@ORIX.com