KRANNERT GRADUATE SCHOOL OF MANAGEMENT

Purdue University
West Lafayette, Indiana

Issues in Server Farm Design for Real Time E-Commerce Transactions

by

Alok Chaturvedi
Mukul Gupta
Sameer Gupta

Issues in Server Farm Design for Real Time E-commerce Transactions·

Alok Chaturvedi
(alok@mgmt.purdue.edu)
Krannert Graduate School of Management
1310 Krannert Building
West Lafayette IN 47906-1310
Telephone: (765) 494-9048
Fax: (765) 494-1526

Mukul Gupta
(mukul_gupta@mgmt.purdue.edu)
Krannert Graduate School of Management
1310 Krannert Building
West Lafayette IN 47906-1310
Telephone: (765) 496-6315
Fax: (765) 494-9658

Sameer Gupta
Krannert Graduate School of Management
1310 Krannert Building
West Lafayette IN 47906-1310
Telephone: (765) 494-2667

Abstract

Server farms constitute the heart of any e-commerce site. This paper evaluates the performance of server farm systems for handling real-time transactions. The paper uses a simulated environment to implement server farms and tests the performance for various design policies and parameters. These design choices include scheduling policy, priority allocation, priority handling, number of machines in server farm, network delays and transaction mix. Results indicate that the prudent policies that schedule all transactions perform very well for low load conditions but fare badly for high load conditions. Simulation also found that some network latency actually improves system performance when the variance in task arrival rates is high. This paper demonstrates that performance of server farm system is contingent on the design choices made during the implementation of the system.

1 BACKGROUND

This paper evaluates the performance of server farm system for handling real-time transactions using a simulation environment. Server farms are computers that are connected together with various networking schemes to evaluate the system to handle large volumes of transactions. Server farms intend to improve the performance of a system by distributing the traffic on network across multiple servers. Load balancer is used for prioritizing and scheduling transactions to individual servers.

E-commerce transactions executed by server farms are critically dependent upon time to execute. Tasks involved in these transactions have to be completed within a stipulated time period. Once the specified deadline elapses, these transactions may lose their values or may even negatively impact the system. These tasks are often referred to as transactions with real-time constraints or real-time transactions for short. For instance, consider an online stock trading system where a trader may demand a trade to be executed within a specified time period or may want the system to execute the trade if the price falls within a certain price window. Trades can also be executed if the trader (or system acting on the instructions of the trader) observes a price differential between two different markets. Window of opportunity during which the price differential exists is small and lasts for a short period of time. Trading operation is useful only if it is executed in this time frame and loses its value once the specified prices cease to exist.

Any system that is responsible for handling the real-time transactions must be capable of handling the real-time requirement imposed by these transactions. For example, one of the design objectives of these e-commerce applications is to maximize the number of transactions serviced within the stipulated time period, but limit the number of transactions serviced that have

already missed their deadlines. It has to be noted that objective of these real-time transaction processing systems is starkly different from the objective of traditional transaction processing systems (TPS). While the conventional TPS focus just on maximizing the average throughput for a set of transactions, the real-time transaction processing systems also have the objective to minimize the number of tardy transactions[1]. This difference can be illustrated using a simple example: Assume two design policies to execute transactions, policy A and policy B. Policy A has an average throughput of 0.6 transactions per unit time and average number of tardy transactions as 15% of all the transactions. Policy set B executes only 0.5 transactions per time period but has only 5% tardy transactions. Conventional transaction processing system would choose policy A and thereby maximize the throughput of the system. By contrast, a real-time transaction processing system would prefer policy B, despite its low throughput, in order to reduce the number of tardy transactions.

Conventional TPS operate with the assumption that all the transactions need to be executed and thereby are required to be committed at some point of time. However, this assumption does not hold true for real-time systems, as some of the transactions have to be deliberately aborted if they fail to meet their required deadlines. A critical operational requirement from these systems is that they detect and abort tardy transactions before they consume excessive computational resources. Failure to detect the tardy transaction in time would result in deterioration of the performance of the system. This might also result in snowball effect. Large number of tardy transactions in the system will consume system resources preventing the system to allocate resources to service other transactions before their deadlines, in turn, making these transactions tardy too.

---

[1] Tardy transactions are the transaction that were not executed before their specified deadline

It might also be dangerous to consider that eliminating all the tardy transaction is a best design policy for a real-time transaction processing system. If the load on the system is low, system might end up using expensive resources just to identify the tardy transactions and eliminate them. In low load conditions, it might be beneficial just to schedule all transactions without worrying about eliminating tardy transactions because systems has enough resources available.

The performance of any real-time system depends upon the design of the control parameters. The design policies and parameters governing a server farm's performance include system load, the transaction scheduling policy, the priority assignments to the transactions, the number of servers or nodes in the farm system, and communication or network delays between the servers. The paper analyzes the impact of these design policies on the performance of server farms. The paper uses a general-purpose simulation system, SimDS [7], a simulation environment for design of distributed systems. This environment provides a mechanism to analyze the policies before they can be implemented in real environment. This research will provide us clues to design of better real time applications.

The rest of the paper is organized as follows: section 2 describes the past research on modeling and evaluation of real-time systems, section 3 introduces design policies and parameters that influence the performance of server farms, section 4 describes the simulation model, section 5 presents the finding of the research, section 6 gives the conclusions, and section 7 presents the implications of the research.

2 LITERATURE

A significant body of research in the distributed systems focuses on real time transactions. However, there is a lack in the research focusing on the use and design of server

farm systems for supporting real-time transactions. Existing distributed system research primarily focuses on developing analytical models of the systems [6, 12, 16, 19, 20, 24, 26]. Other researchers have focused on analyzing different mechanism for operation of real-time systems under different environmental conditions [8, 12, 21, 22, 23]. They use these models to analyze the performance of distributed systems. These researchers also proposed certain design policies that control the performance of the systems. We have utilized some of these design policies in our simulation. Some researchers have performed simulations to analyze the performance of real-time systems for specific design parameters [1, 2, 3, 4]. However, none of the simulation-based studies have attempted to simulate the utilization of server farm systems for real-time transactions.

Existing server farm literature focuses primarily on analyzing the performance of server farm systems for web hosting services [31]. Researchers also have focused on developing load-balancing strategies for these server farm systems [9, 32]. Load balancing software is the heart of any server farm. Load balancer tracks the demand for processing power from different machines, prioritizes the tasks, and schedules and reschedules tasks to individual machines. Several vendors are involved in development of commercial server farm systems. These include Blackstone Technology Group, Compaq and Sun Microsystems.

3   DESIGN POLICIES

As mentioned in the earlier section, the performance of a server farm depends on several policy parameters specified during the design of the system. In this section, we present few of the design policies that can potentially impact the performance of server farms. In the later sections, we analyze the performance of server farms for a subset of these policies.

The first set of policies, implemented by a load balancer, governs the scheduling of the transactions to machines in a server farm. Load balancer tracks demands for processing power from different machines, prioritizes the tasks and schedules or reschedules them. Different transactions have different value associated with completion of the transaction or have different opportunity costs for missing the deadlines. Some system critical transaction might require execution at all costs. Other transactions might be aborted easily without much loss to the value of the system as a whole. Two mechanisms are involved in scheduling of transactions. First mechanism assigns priorities to incoming transactions [8, 13, 25, 27]. Second mechanism defines how the load balancer is going to schedule transactions with different priorities.

- *Priority Value Allocation*: Three different options for priority allocation can be utilized to assign priorities to transactions

  o *FIFO Policy*: The policy assigns the priorities based on the first come first serve basis i.e. the first transaction arriving at the system receives the highest priority while the last transaction arriving at the system receives the lowest priority.

  o *Deadline Policy*: This policy assigns priorities based on transaction deadlines by assigning higher priority to transactions with earlier deadlines.

  o *Slack Policy*: This policy assigns priorities based on the slack time for the transactions. The slack time refers to the time differential between the transaction deadline and the estimated processing time for the transaction. Transaction with the least slack time is assigned the highest priority

- *Scheduling Policies*: Once a priority has been assigned to a transaction, the load balancer needs to allocate the processor time for the transaction [6, 14]. If any of the machines in the server farms is idle than the transaction is scheduled immediately to

the idle server. If none of the machines are available and all of them are servicing higher priority transactions then the arriving transaction has to wait for the arriving transaction to complete before it can be scheduled. In this case, the transaction is just placed in the queue. But if the arriving transaction has a higher priority than all the transaction being serviced at that moment, the scheduler implements one of the following policies to resolve the contention of the arriving transaction to use the system.

- o *Non-preemptive Policy*: In this policy, all the current transaction are allowed to complete before any new transaction is scheduled. Once, any of the machine becomes idle, the first transaction with the highest priority in the queue is scheduled.

- o *Preemptive Policy*: Under this policy, as soon as a high priority transaction arrives, the transaction with the lowest priority is aborted immediately. The arriving transaction is scheduled immediately to the server that was servicing the aborted transaction. The current transaction maintains its existing priority value implying that if another high priority transaction arrives before any machine becomes available the arriving transaction will be scheduled ahead of the aborted transaction.

- o *Promoting Preemptive Policy*: This policy is similar to the preemptive policy. However, in this case the aborted transaction assumes the priority of the new transaction. This ensures that the transaction receives a fair chance of getting executed and does not have to spent most of its time getting in and out of the queue.

o *Conditional Non-preemptive Policy*: This policy allows the current transaction to complete if the system estimates that the arriving high priority transaction can afford to wait for one of the current transactions to be completed. The system computes the slack time of the arriving transaction, and if this slack time is greater than the estimated processing time for any of the current transactions, it allows all the current transactions to execute.

Another design issue that a load balancer must consider while scheduling the transactions is the handling of the 'overloading transactions' [1, 2, 14]. The overloading transactions are defined as transactions that are unlikely to be completed before their deadlines. Three strategies to manage the overloading transactions are presented here:

- *Schedule All Policy*: This policy completely ignores the overload issue and schedules all transactions irrespective of their likelihood of completion.

- *Tardy Policy*: Tardy policy focuses on aborting all transactions whose deadlines have elapsed. Policy ensures that all tardy transactions are removed from the queue at the earliest opportunity. This policy is not concerned with transactions that are estimated to miss their deadlines but is only concerned with transactions that have already missed their deadlines.

- *Feasible Policy*: This policy aborts all transactions that are unlikely to be completed before their respective deadlines. Load balancer achieves this by computing the estimated slack times for all transactions and aborting the transactions that have negative slack times. Practical implementation of this policy actually involves some overhead costs in computing the slack times for all transactions. However, our simulation does not model these overheads.

Each server in a server farm must have access to the appropriate data to execute the transactions. When several machines serve the same content, following policies for data fragmentation and replication can be implemented [12, 15, 19]:

- *No Replication Policy (Shared Storage)*: This policy assumes that only one copy of data exists throughout the server farm. This policy gains relevance if application requires frequent data updates, network is relatively failure safe, and network delays are minimal.

- *Full Replication Policy (Replicated Storage)*: In this policy, the system replicates and stores all the data at each machine. This policy is viable where the response time is critical, the network is somewhat unreliable, and network delays are significant.

- *Partial Replication Policy (Hybrid Solution)*: This policy implementation involves a mix of no-replication and full-replication policies. Different number of replicas for different data sets may be kept at several sites. Determination of the number of copies for a particular data set depends on the frequency of access needed to the data set at each site, network delays and response time required by the system.

Simultaneous access to data source gives rise to issues related to consistency and serializability of transactions [5, 12, 20]. Two widely accepted concurrency control policies that could be implemented in server farms are:

- *Locking Based Policy*: Under this policy, each transaction requests and obtains access rights (called locks) prior to accessing data source. If the transaction is requesting to modify the data in some form, the locks are provided on exclusive basis. But if the transaction is interested just in reading the data, the locks can be shared by several

transactions at the same time. The locks owned by a transaction are released once the transaction is completed.

- *Optimistic Policy*: Under this policy, the transactions are permitted to execute without any kind of locking mechanism. However, once the transaction is complete and is ready to commit, the system ensures that the transaction does not conflict with any of the uncommitted transactions present in the system. If no conflict is present then the transaction is committed otherwise some of the conflicting transactions are aborted to ensure serializability.

A locking conflict may arise when a server or transaction tries to obtain a lock to a transaction that is already locked. Database systems use different resolution policies to tackle such situations [11, 18]. Two popularly implemented conflict resolution policies are:

- *Blocking Policy*: Blocking policy suspends the requesting transaction for duration that is equal to completion time for an average transaction.

- *Restart Policy*: Restart policy terminates the transaction and informs the originating site. Originating site than proceeds to resubmit the lock request.

4   SIMULATION MODEL

This paper simulates a server farm configuration as shown in Figure 1. Each simulated server farm configuration is assumed to consist of several machines (or servers or nodes). Each machine is assumed to have both computing and data storage capabilities. For the purpose of the simulation, we assume that all the servers are fail safe i.e. they are always up and running. We do not address the issue of load balancing in case of node failures. Handling redistribution of load in case of failures is left to be addressed in the future research.
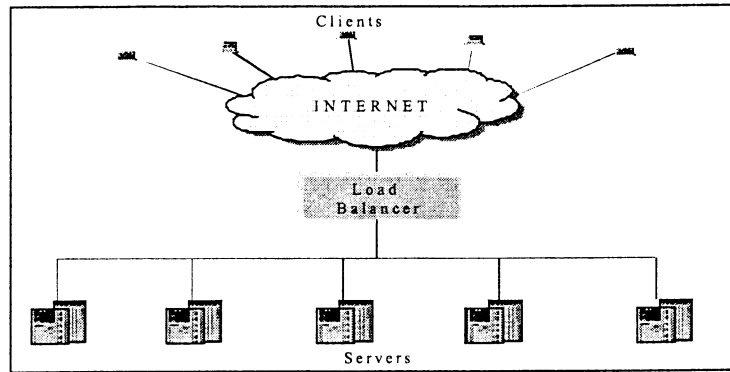
Figure 1: Server Farm Architecture

Simulation assumes relational database model for data storage. Data is logically arranged as tables, and tables may or may not be replicated on different servers in the server farm depending on the data distribution and replication policies. For the purpose of simulation, the granularity of data replication is assumed to be one table [10]. Each table, however, has a designated primary site that is kept fixed for the period of the simulation. This primary site is responsible for performing all serializability and concurrency tests on the data contained in the table. In the case of centralized concurrency control, our simulation assumes that site 1 is always the locking site.

Load balancer distributes the incoming transactions to individual machines based on the scheduling policy used. Each transaction has its associated releases time and a deadline before which the transaction needs to be executed. The simulation assumes deadlines to be known a priori. The simulation also assumes that each transaction has access privileges to all data sets maintained at all servers. The issue of data access and security are not being considered in this simulation [17, 28]. It is also assumed that each server has the ability to determine the entire read and write requirements for the arriving transaction and distributes the sub-transactions to the necessary nodes. This server is also responsible for sending lock requests to the site maintaining

the locks for these sub-transactions. All sub-transactions have to wait for the receipt of the locks before being scheduled for execution.

Once the sub-transactions have been executed, all the results are returned to originating machine for computations. This machine computes the final results and initiates the commit process. The transaction is said to be complete once the results have been computed but before the commit process is initiated. This assumption is valid because we assume that all servers and networks are operational at all times.

If one of the sites executing the sub-transactions decides to abort a transaction, it has to inform the originating site, which then initiates and coordinates the abort process between all sites. Each aborted transaction has to release all locks that it had acquired. Thereby, the cost of rolling back a transaction is same as the cost of acquiring the locks.

To maintain the integrity of the process, 'atomicity of broadcasts' has to be preserved, i.e., the system must ensure that all transactions are received in the correct order. This is implemented by maintaining a separate logical network queue for each pair of sites. Each new message in this logical queue is assigned a random delay greater than the delays associated with transactions already in the queue.

We have used two parameters to measure the performance of the system under various policy combinations. These parameters are

- *Number of OK transactions*: Number of OK transactions are all the transactions that have been completed before their respective deadlines expired.

- *Throughput*: Throughput is defined as total number of transactions serviced per unit of simulation time. (Throughput is presented in 1000's in the results)

Experiment design consisted of 20 different sets of experiments with different seeds. In each experiment, the systems monitors 600 incoming transactions out of which first 200 are ignored to enable the system to attain a steady state. The simulation clock is reset after 200 transactions are completed.

5 RESULTS AND DISCUSSION

The performance of a server farm is governed by several independent design policies described in earlier section. However, evaluating system performance for all possible combinations of the design parameters requires an exponential amount of time. Thereby, for this simulation we adopted a realistic strategy to examine the system performance for a priori selected combinations of design parameters. We also tried to understand the impact of individual design choices and attempted to discern the underlying reasons for the observed behaviors.

5.1 Analysis of Overloading Transaction Scheduling Policy on the Performance of a Server Farm System

The system load is implemented as the arrival rate of transactions at each node. Figure 2 & Figure 3 indicate the impact of system load on the performance of a two machine server farm system under disparate overloading transaction handling policies implemented by the load balancer.

The results indicate that the system performance is stable when the load balancer follows 'Feasible' or 'No Tardy' scheduling policies. However, when 'Schedule All' policy is followed system performance deteriorates considerably for high loads. Throughput of system exhibits a 'bell shaped curve' for 'Schedule All' policy. This indicates that system has an optimal arrival rate for which system achieves a maximum throughput. The arrival rate governs the system

performance before this optimal and the service times govern the system performance after the optimal. The number of OK transactions decrease for all the policies with increase in system load but number of OK transactions reduce drastically after the optimal arrival rate in the case of 'Schedule All' policy.
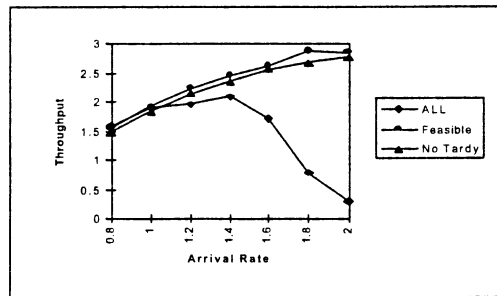
Figure 2: System throughput for two machine server farm under disparate overloading transaction handling policies
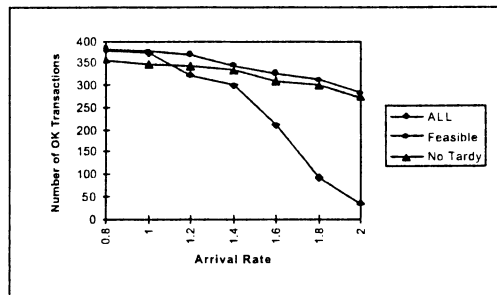
Figure 3: Number of OK transactions serviced by two machine server farm under disparate overloading transaction handling policies

The system performance does not deteriorate with arrival rate for 'No Tardy' and 'Feasible' policies due to the fact that both of these policies focus on eliminating the tardy or potential tardy transactions, thereby, reducing the number of transactions waiting to be serviced at any point of time. The number of transactions in the queue does not explode with increase in arrival rate and system does not reach critical state. As Figure 3 indicates, only few transactions miss their deadlines under these policies. By comparison, in 'Schedule All' policy the queue sizes keep on increasing due to excessive waiting time of the transactions. Figure 4 shows the

average queue lengths in the CPU for each node under 'Schedule All' policy. Average queue

lengths increase sharply toward the upper end of the arrival rate spectrum. Increase in queue

length causes the increase in waiting time for arriving transactions causing some of these

transactions to miss their deadlines, thereby; number of OK transactions serviced decreases

rapidly.

Figure 4 also indicates that queue lengths at the locking site are greater than the queues at

the other site and rise more steeply with the increase in the system load. Therefore, the CPU

availability at the locking site is one of the bottlenecks in distributed transaction processing
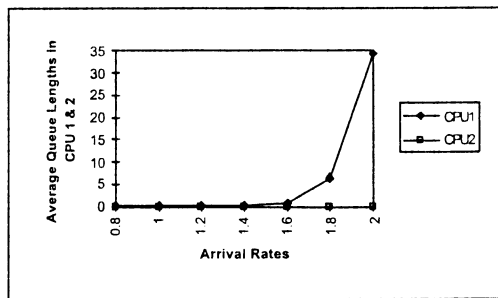
systems.

Figure 4: Average queue lengths in CPU for 'Schedule All' Policy

Figure 5 & Figure 6 show the impact of scheduling policies on the system performance

for a single server system. The impact of scheduling policies on the performance is much more

clear for such a system than with more complex systems.
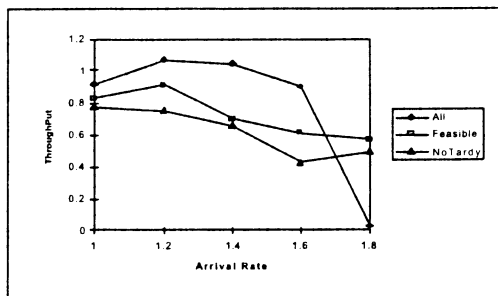
Figure 5: System throughput for one node system under different scheduling policies

For lightly loaded systems (very few arrivals per second), the 'Schedule All' policy performs better than the other two policies. But when the system load increases, 'Feasible' and 'No Tardy' policies outperform 'Schedule All' policy. Another interesting observation is that 'Feasible' policy always performs better than 'No Tardy' policy. As pointed out earlier, the deterioration in the performance of 'Schedule All' policy is related to the waiting time of transactions in process queues at all nodes. The distinction between the other two policies is more subtle. 'Feasible' policy aborts more transaction than 'No Tardy' policy.
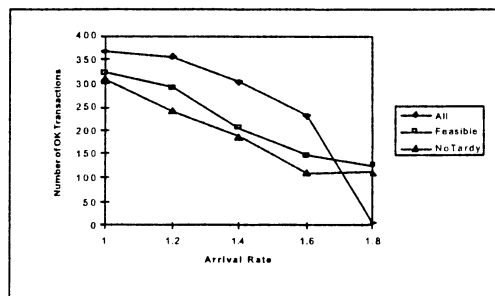
Figure 6: Number of OK transactions for a one node system under different scheduling policies

Figure 7 & Figure 8 analyze the system performance for a four server system. For low arrival rates, the performance of a four machine server farm system exhibits the same behavior as that of a one or two machine server farm. However, for high arrival rates, the system performance does not suffer and throughput increases monotonically with arrival rates. This behavior can be attributed to the fact that system did not attain its optimal. This finding indicates that adding more machines to a server farm increases the overall throughput of the system. The number of OK transactions show a negative correlation to arrival rates.

These results give us good indication to the scheduling policies that can be followed by load balancer to distribute the load to the machines in a server farm. 'No Tardy' and 'Feasible' policies provide overall better performance but 'Schedule All' policy outperforms them under

low load condition. A load balancer encounters both periods of low as well as high load conditions. Thus, the best options for a load balancer is to implement an intelligent mechanism for aborting transactions. This aborting mechanism should not terminate any transactions prematurely at low arrival rates but should increase the intensity of aborting transactions as system load increases to prevent the system from being overloaded. This amounts to the fact that system should use 'Feasible' or 'No Tardy' policy for high system load conditions and should prefer 'Schedule All' policy for low system load.
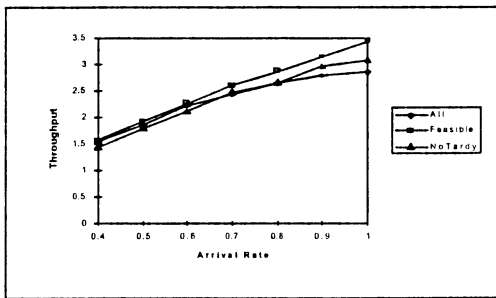
Figure 7: System throughput for four node server farm under different scheduling policies
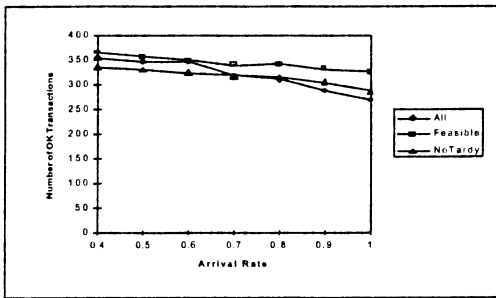
Figure 8: Number of OK transactions for a four node server farm under different scheduling policies

5.2 Analysis of impact of Priority Assignments Policies on the performance of the system

Each transaction that arrives at arrives at the server farm system is assigned a priority. This priority value is used by the scheduler to schedule transactions to the processor. Once a

process becomes free, it selects the next transaction to be serviced based on the priorities of the transactions waiting to be serviced.
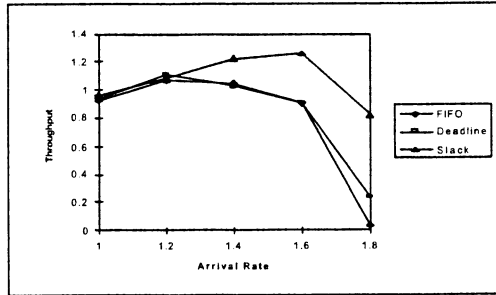
Figure 9: System throughput for a one node system under different priority handling policies

In this section, we analyze the performance of various priority handling policies for 'Schedule All' scheduling policy. Figure 9 & Figure 10 show the impact of priority handling policies on the system performance for a one node system.
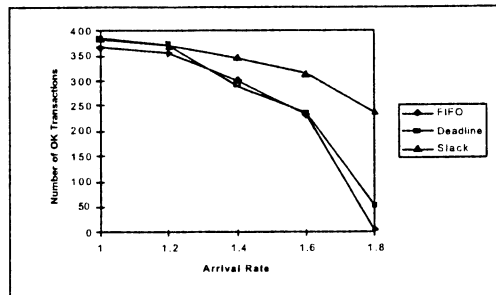
Figure 10: Number of OK transactions serviced by a one node system for different priority handling policies

Figure 11 & Figure 12 show the impact of various priority handling policies on the performance of a two node server farm. It can be seen from the figures that the priority assignment policies do not determine the system performance under low load conditions. This result corresponds to the intuition. For low arrival rates the number of transaction waiting in the queue is small (often zero), thereby, eliminating the impact of priority policies. This result is also supported by Figure 4 that illustrates that CPU queues in all nodes are very small under low

19

load conditions. Priority policies gain significance once there are multiple transactions waiting in the queue.
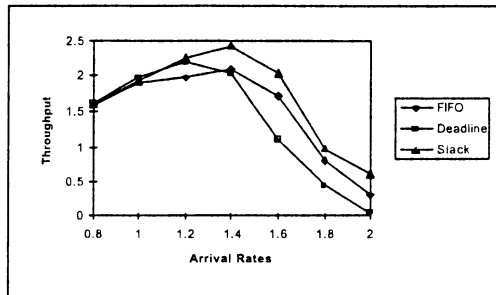
Figure 11: System throughput for a two node server farm under different priority handling policies
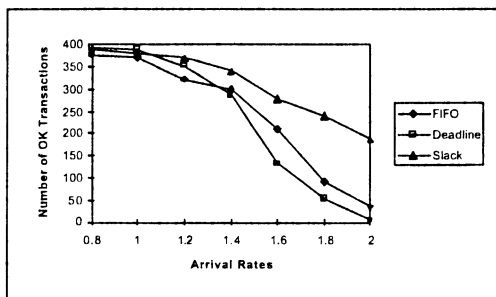
Figure 12: Number of OK transactions serviced by a two node server farm for different priority handling policies

When the system load increases, the 'Slack' policy significantly outperforms both 'FIFO' as well as 'Deadline' policy. 'FIFO' policy performance the worst, which justifies the intuition as this policy ignores the deadlines and schedules the transactions as they arrive. This causes transactions with short deadlines to miss their deadlines adversely affecting the system performance. The 'Slack' policy estimates the 'slack time' by estimating the service time for the transaction. The performance of the 'Slack' policy is dependent of this estimation of the service time of transactions. If the slack estimates are low, the 'Slack' policy will behave similar to 'Deadline' policy by scheduling the transactions with earlier deadlines before other

transactions. However, if the slack estimates are infinitely large, 'Slack' policy mimics 'FIFO'

by servicing transaction on first come first serve basis.

It has to be noted that for the purpose of our experiments, we assume that the deadlines

for all transactions are known a priori. In actual systems this is not the case. In real life,

systems will have to estimate the deadlines of the transaction and make the policy decisions

based on those deadlines. Correctness of these estimates will play a significant role in

determining the performance of the system. However, investigation into deadline estimation

methodologies is not the focus of this paper.

5.3 Analysis of Impact of Concurrency Conflict Resolution Policies

When a transaction tries to obtain locks on a data item, concurrency conflict arises if the

data item is already locked by another transaction. Figure 13 & Figure 14 present the impact of

'Blocking' and 'Restart' conflict resolution policies for a two-node system implementing

'Schedule All' policy.
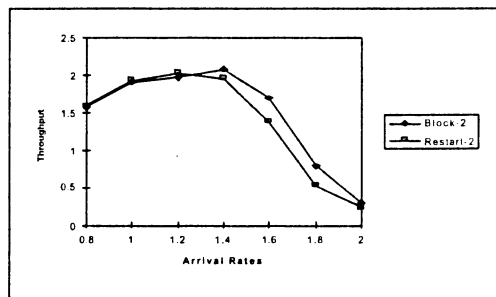
Figure 13: System throughput for a two node server farm under different conflict resolution
policies

The figures indicated an unexpected but interesting result. The system performance for a two node

system is not dependent on the conflict resolution policies. We will have to examine the assumption

closely to analyze this result. The simulation assumes that all data items are equally likely to be accessed

by any given transaction. When the system load is low, the number of outstanding transactions is few and

the probability of conflict is small. This precludes the need for conflict resolutions, thereby; the concurrency conflict resolution policies have no impact on the system performance under low load conditions.
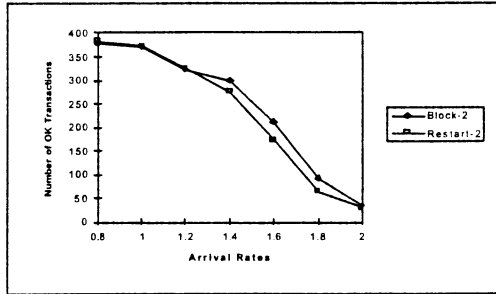
Figure 14: Number of OK transactions serviced by a two node server farm under two different conflict resolution policies

At high loads, system performance is governed by other factors, particularly the waiting time in queue for the transactions. These factors mask any impact that the conflict resolution policies have on the performance of the system, hence making it difficult to isolate the impact of conflict resolution policies on the system performance. Even for intermediate load ranges, there is little difference in the performance of the system for both conflict resolution policies, although 'Blocking' policy seems to perform slightly better than the 'Restart' policy. This differential can be explained by the fact that immediately restarting a transaction tends to flood the system, thereby, reducing system performance. However, this difference in performance is not significant.

We recommend that a real-time distributed transaction processing system can use any of the two conflict resolution policies because the system performance is dominated by scheduling policies and not impacted much by the conflict resolution policies. A system can make the choice of conflict resolution policy based on ease of implementation of the policy.

5.4 Impact of Network Delays on the Performance of the System

In the preliminary set of experiments, the performance of the system was observed for no network delays. But in real world, transactions are subjected to network delays and these network delays tend to vary significantly depending on the message size being transmitted. Figure 15 & Figure 16 show the performance of the system for a four node server farm under None, Small, or Large network delay conditions. For the experiments, the message delays were derived from identical distributions.
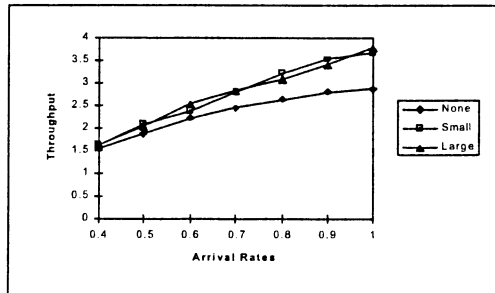
Figure 15: Impact network delays on the throughput of a four node server farm
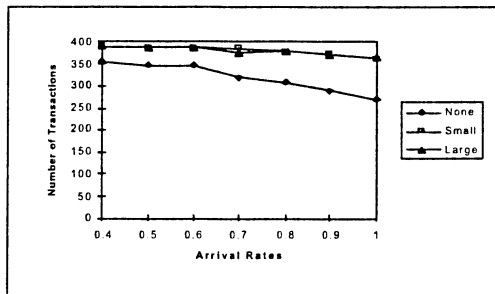
Figure 16: Impact of network delays on the number of OK transactions serviced by a four node server farm

These figures uncover a very interesting finding. The system performance actually improves with network delays i.e. the system performed better when the transactions were associated with large network delays. Furthermore, the gap between the performance levels for low and high network delays actually increased with the increase in the arrival rate for incoming

transactions. One possible explanation for this anomalous behavior could be that the network delays actually offset the system load by reducing the size of queues in the system. However, the performance of the system needs to be analyzed further before any conclusion can be drawn.

We conducted additional experiments to analyze this unexpected result further. Figure 17 & Figure 18 present the impact of the various network delays for a four node system with high arrival rate and segmented by scheduling policies: 'Schedule All', 'Feasible' and 'No Tardy'. The network delays for these experiments were varied from 10^{-4} to 10^{-2} seconds per transaction.
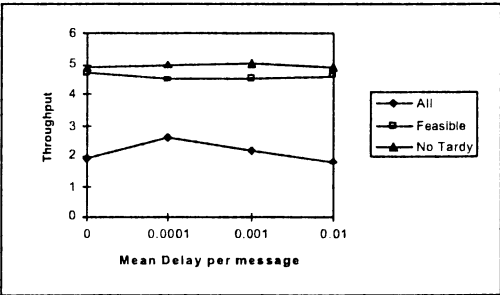
Figure 17: System throughput for a four node server farm under different network delays segmented by scheduling policies
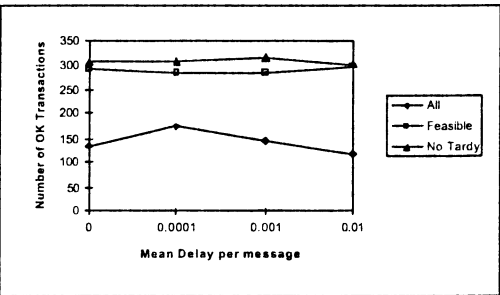
Figure 18: Number of OK transactions serviced by a four node server farm for different network delays segmented by scheduling policies

The figures indicate that the system performance is steady for networks delays under 'Feasible' and 'No Tardy' policies. But, for 'Schedule All' policy system performance follows a 'bell shaped' curve i.e. the system performance improves initially with the network delay but the

24

performance declines for larger delays. These finding suggest that network delay offsets the load

on the system processes. A number of factors can be responsible for this trend in system

performance. This behavior can also be associated with the instability in the system. We

further analyze the performance of system to isolate the effects of 'temporary instability'.

Temporary instability in the system can be caused by high variance in the arrival rates.

The variance in the arrival rate can cause a temporary accumulation of transaction in the process

queue. Since each transaction has a deadline, it has a time frame within which it needs to be

completed, thus, temporary instability can cause a significant reduction in the system

performance. Figure 19 & Figure 20 analyze the impact of variance in interarrival times on the

system performance under 'Schedule All' policy. During experimentation, interarrival rates

wire generated from uniform distribution having the same mean (1.6 transactions per second) but

different variance (varying from 0.00033 to 0.208).
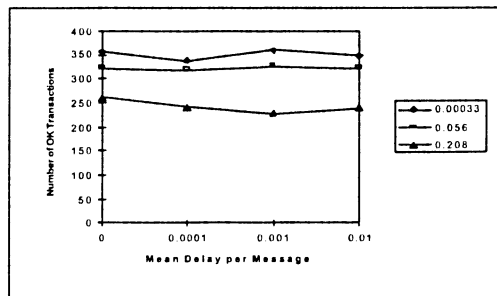
Figure 19: System throughput for a four node server farm for different network delays
segmented by variance in the interarrival times

The findings indicate a significant negative impact of delay variance on the system

performance. Increasing variability in the arrival times is directly proportional to the increasing

variability in queue lengths, thereby, creating 'temporary instability'. This causes the

transactions to be delayed beyond deadlines, which reduces the system performance. It seems

that the network delay actually reduces the variability in the queue length by spreading out the

arrival of transactions. This causes the reduction in the temporary instability, hence improving

the performance of the system for high network delays.
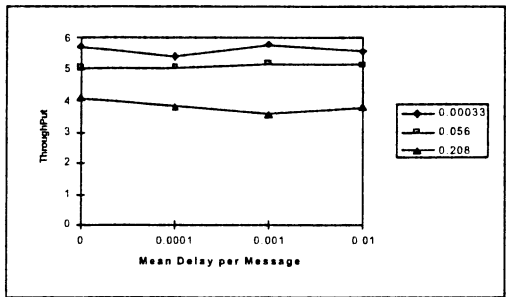
Figure 20: Number of OK transactions serviced by a four node server farm under different

network delays and segmented by variance in the interarrival times

We also analyzed how the network delay impacts the system performance for different

types of transactions. Impact of delays on read-only transactions is restricted to delays in lock

and unlock requests since the read-only transactions are processed locally. Update transactions,

on the other hand, are more likely to be effected by the network delays. We examined the impact

of delays on the type of transactions by varying the incoming mix of transactions. The

transaction mix is defined by the fraction of update transactions in the transaction mix. Figure

21 & Figure 22 illustrate the impact of mix of incoming transactions on the system performance.
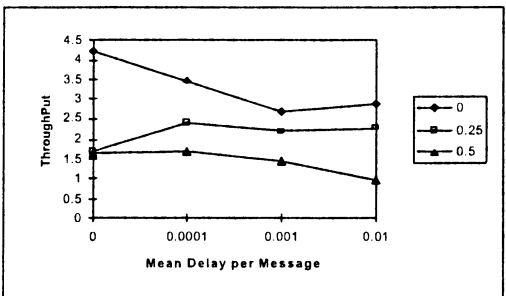
Figure 21: System throughput for update transactions in the mix varying from

The figures indicate that the system performance is better for read-only transactions,

however, the system performance suffers for high network delays. The gap between the

performance levels for no update transactions and 50% update transactions widens for high

network delays. The observations support the intuition that network delays impact different

types of transactions differently and allow some transactions to be completed at the expense of

others.
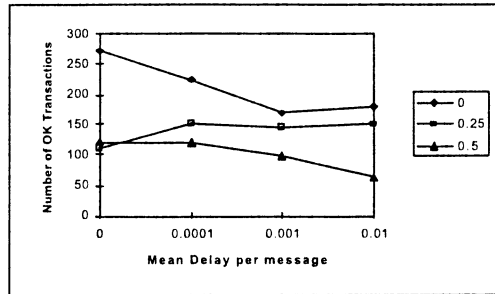
*Figure 22: Number of OK transactions serviced by the system for update transactions in the
mix varying from 0 to 50 %*

5.5   Scalability

So far we have illustrated the performance of real-time distributed transaction processing

systems for several design parameters. But one other important design parameter is the number

of network nodes itself. Figure 23 & Figure 24 depict the performance of the system for one,

two, and eight node systems.

The figures indicate that the number of transactions serviced by the system before the

respective deadlines is almost same for all systems at low loads. However, the throughput of the

system varies widely depending on the number of nodes present in the system. The system with

higher number of nodes yields a significantly higher throughput. This is not unexpected as a

higher node system also associated higher computing resources. However, at high system loads,

the number of OK transactions serviced by the eight-node system is much less than those for the

corresponding one or two node systems. The throughput of the eight-node system undergoes

drastic deterioration for high system loads as compared to less complex systems. This steep

decline may be due to high overheads required to maintain the data integrity for more complex systems. Each transaction spawns a number of sub-transactions to perform lock, compute, commit, and unlock operations. Some of these sub-transactions might be required to be sent to all data nodes and may be needed to execute at all sites. In complex systems, the number of sub-transactions may overwhelm the system causing a sharp degradation in the performance of the system.
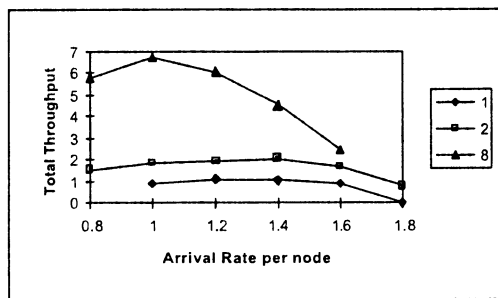
Figure 23: System throughput for one, two, and eight node server farm
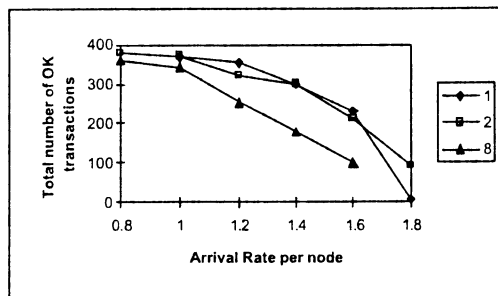
Figure 24: Number of OK transactions serviced by one, two, and eight node server farm

Another useful metric to evaluate the system performance is average system throughput as compared to the total system throughput. Figure 25 presents the average throughputs for one, two, and eight machine server farm.

The throughput per node for the eight-node system is worse than that for less complex systems. This is in contrast to the total throughput that was more for the eight-node system. The difference in average throughput for eight-node system and less complex systems increases

as the system load increases. Further research needs to be carried out to analyze the metrics that

capture the economic impact of the complex systems. This research is beyond the scope of this

paper.
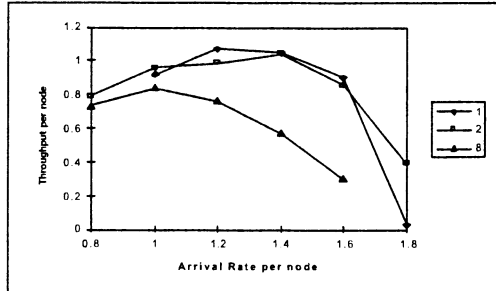
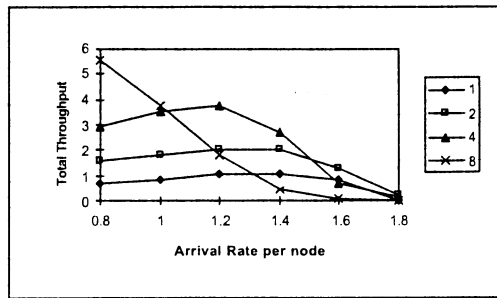*Figure 25: Average throughput for one, two, and eight node server farm*
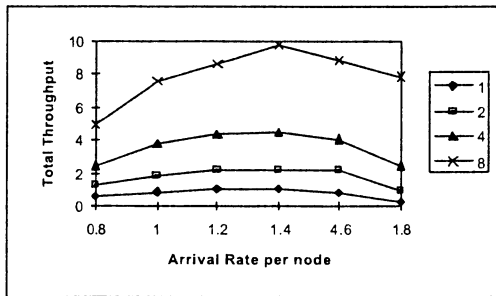
Figure 26: Total throughput of the system for one, two, four, and eight node server farm with
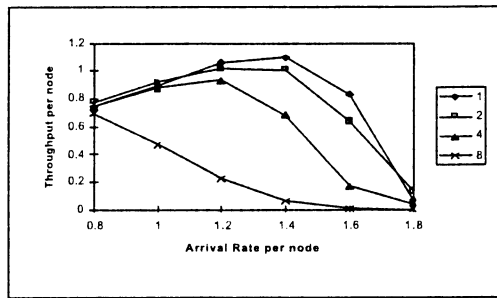
0% update transactions in the mix
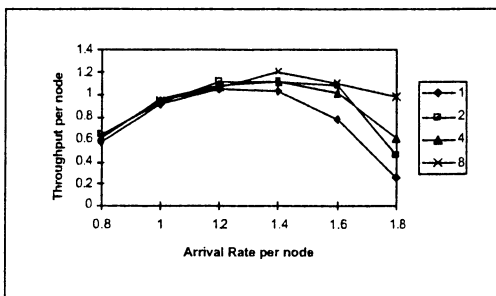
Figure 27: Average throughput of the system for one, two, four, and eight node server farm

with 0% update transactions in the mix

Another interesting thing to evaluate is how the performance of the system is affected by

the complexity of the system for different kinds of transactions. Figure 26 through Figure 33

elucidate the total throughputs and average throughputs for one, two, four, and eight node server

farm system for different transaction mixes.
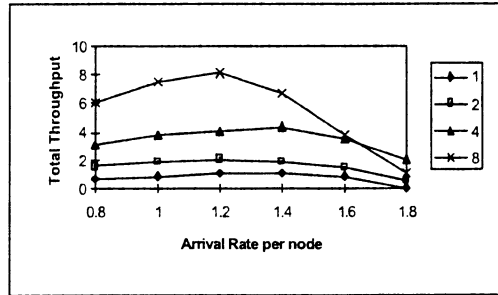
Figure 28: Total throughput of the system for one, two, four, and eight node server farm with

12.5% update transactions in the mix
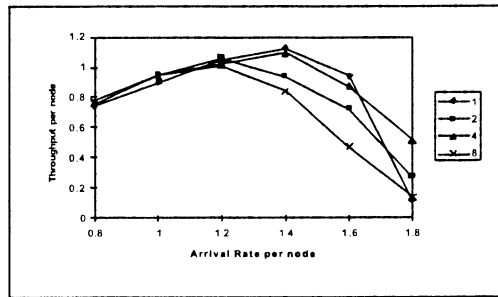
Figure 29: Average throughput of the system for one, two, four, and eight node server farm

with 12.5% update transactions in the mix
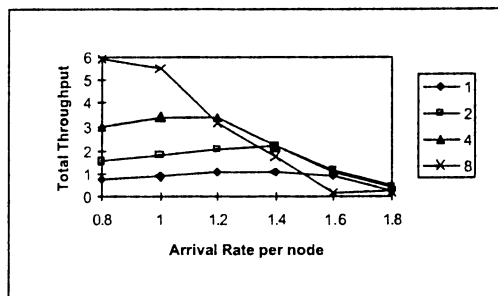
Figure 30: Total throughput of the system for one, two, four, and eight node server farm with

37.5% update transactions in the mix
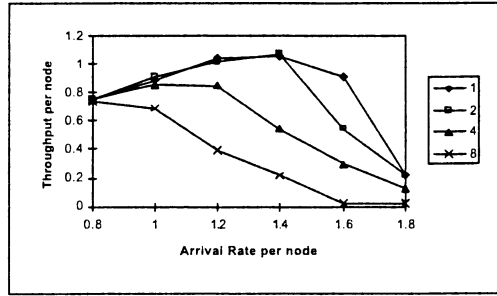
Figure 31: Average throughput of the system for one, two, four, and eight node server farm with 37.5% update transactions in the mix

Figure 32: Total throughput of the system for one, two, four, and eight node server farm with 50% update transactions in the mix

Figure 33: Average throughput of the system for one, two, four, and eight node server farm with 50% update transactions in the mix

The figures show some interesting trends in the behaviors of the system for different transaction mixes. The performance of a distributed system is scalable if all transactions are read-only transactions. The average throughput remains same in almost all load conditions for one, two, four, and eight node server farm systems for read only transactions. Thus, the

throughput for an eight node server farm is almost four times the throughput for a two node system amounting to a linear scalability. However, if the transaction load is near the unstable region, the system performance suffers faster for more complex systems.

When the fraction of update transaction is 50% in the transaction mix, the total throughput for complex systems is better than less complex systems. However, the total throughput deteriorates rapidly in complex systems as system load increases. Thereby, for high load conditions, the total throughput for less complex systems is actually better than that of more complex systems. In the intermediate range of transaction mixes, the degradation in performance exhibits a monotonic behavior, i.e., the degradation in performance with system load in complex systems is quicker than that in simple system. Also, this rate of degradation increases with the increase in fraction of update transactions in the transaction mix.

The above findings suggest that transaction mix should govern the design of a real-time distributed transaction processing system. If the system is to be designed for more read only operations, it is a good idea to create large number of nodes in the system for faster processing and better system throughput. This justifies the use of server farm systems for web viewing applications. However, if the system is responsible for high number of update transactions, the update capabilities should be limited to small number of nodes.

6 CONCLUSIONS

The results of this research provide insights into design of server farm systems for handling real time e-commerce transactions. Experiments demonstrate that system performance, defined by number of successful transactions serviced and the system throughput, is dependent on several factors. Perhaps, most critical of these factors is the scheduling policy that governs the handling of transactions arriving at the system. The system performance

degrades sharply with the system load for a prudent scheduling policy where all transactions are scheduled. At high loads, policies that tend to eliminate transactions that have already missed deadlines or are expected to miss deadlines performed much better. Thereby, we need an intelligent system that schedules all transactions at low system loads and eliminates all 'tardy' transactions at high loads. The success of this system is critically dependent on the estimation of the time required to complete a transaction.

An interesting finding of this research was that the concurrency conflict resolution schemes did not affect the system performance for any system load as the system performance is dominated by the waiting time in queue under all conflict resolution schemes. Systems with high variance in arrival rate actually benefit from small to medium network delays. Thereby, if the transaction arrival rate has high variance, spreading out the transaction would help the system performance. Performance for highly complex systems degrades rapidly if there is high fraction of update transactions arriving at the system. However, the system throughput increases for complex systems servicing only read only transactions. We recommend spreading the read capabilities across all the nodes in the distributed system and limiting the update capabilities only to few nodes.

In this research, we tested only a subset of design policies that can be implemented by a real time distributed transaction processing system. The design policies not covered in this research can be tested in future research. We also did not determine an economic measure to evaluate the value of a complex system. In this research we also assumed the transaction deadlines to be known a priori. A probabilistic model can be developed in future research to relax this assumption. We also did not consider partial mix of read and write policies in the

transaction mix. Partial replication of read-write policies can give us a better insight in the design of the complexity of the system.

7 DESIGN IMPLICATIONS

This section presents two major design implications inferred from the study. First implication pertains to the network delays experienced by transactions and second concerns the scheduling of overloading transactions by the load balancer.

- *Implication 1*: It was noticed that a small amount of network delay actually enhanced the performance of the system by reducing the variance in the arrival of the transactions. However, high network delays deteriorated the system performance. This implies that a hierarchical structure is more suitable for a server farm system (Figure 34). This structure would reduce large network delays. However, transactions will still face small network delays. The performance evaluation for this hierarchical structure will be undertaken in future research.
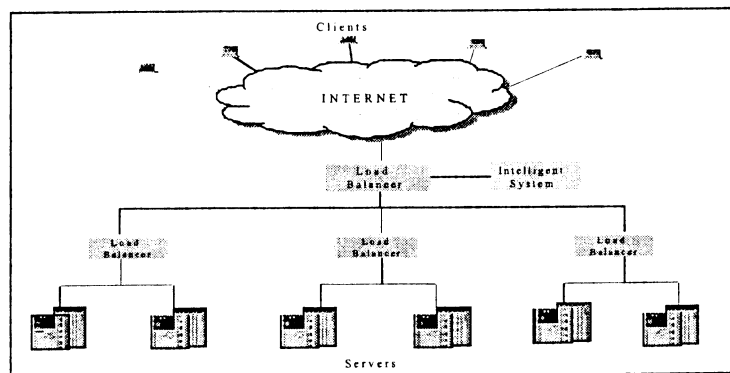
Figure 34: Hierarchical Server Farm Architecture

- *Implication 2*: 'Schedule All' policy performed better than 'Feasible' or 'No Tardy' policy under high load conditions. However, the performance of the system reduced drastically for 'Schedule All' policy with increase in system load. We propose an

34

intelligent system to be used with the load balancer (Figure 34). This system would

sense the load and decide which scheduling policy to be use. The implementation and

evaluation of this intelligent system is scheduled for future research.

8 REFERENCES

[1] Abott, R.K and Garcia-Molina, H. Scheduling real-time transactions. ACM SIGMOD Rec. (March 1988), 71-81.

[2] Abott, R.K and Garcia-Molina, H. Scheduling real-time transactions: A performance evaluation. Proceedings of the 14th VLDB Conference (Los Angeles, Aug. 29-Sept. 1, 1988), 1-12.

[3] Abott, R.K and Garcia-Molina, H. Scheduling I/O transactions with deadlines: A performance evaluation. IEEE Real-Time System Symposium, (Dec. 1990), 113-124.

[4] Abott, R.K and Garcia-Molina, H. Scheduling real-time transactions: A performance evaluation. ACM Transactions on database systems, vol. 17, no. 3, (September 1992), 513-560.

[5] Adya, A., Gruber, R., Liskov, B. and Mahewhwari, U. Efficient Optimistic Concurrency Control Using Loosely Synchronized Clocks, Proceedings of the ACM SIGMOID International Conference on the Management of Data, May 1995

[6] Chaturvedi A.R., Choubey A.K, and Roan J Scheduling the allocation of data fragments in a distributed database environment : A machine learning approach. IEEE Transactions on Engineering Management, vol. 41, No. 2, (May 1994), 194-207.

[7] Chaturvedi, A. R., Gupta, Samir, and Bandyopadhyay, S. "SimDS: A Simulation Environment for the Design of Distributed Database Systems", DATABASE, vol 29(3), (Summer 1998), 65-81

[8] Carey, M., Jauhari, R. and Livny, M. Priority in DBMS resource scheduling. Proceedings of the 15th VLDB conference, 397-410.

[9] Cherkasova, L., "FLEX: Load Balancing and Management Strategy for Scalable Web Hosting Service", Proceedings of the Fifth International Symposium on Computers and Communications (ISCC'00), Antibes, France, July 3-7, 2000

[10] Dayal, U., Blaustein, B. et. al. The HiPAC project: Combining active databases and timing constraints. ACM SIGMOD Rec.(March 1988), 51-70.

[11] Davidson, S., Lee, I. and Wolfe, V. A protocol for timed atomic commitment. IEEE conference on distributed computing systems, Newport Beach, CA, Jun 1989, 199-206.

[12] Gavish B., and Suh M.W. Configuration of fully replicated distributed database system over wide area networks. Annals of Operation Research, vol. 36, (1992), 167-192.

[13] Huang, J, Stankovic, J. A., Ramamritham, K., Towsley, D., and Purimetla, B., .. Priority Inheritance in Soft Real-Time Databases, The Journal of Real-Time System, vol. 4, 1992, 243-268

[14] Hong, D., Johnson, T. and Chakravarthy, S. Rea-Time Transaction Scheduling: A Cost Conscious Approach, SIGMOD Conference, 1993, 197-206

[15] Hsiao H., and Dewitt D. Performance study of 3 high availability data replication sttrategies. Distributed and Parallel Databases, vol. 1, No. 1, (Jan. 1993), 53-80.

[16] Lee H., and Liu Sheng O.R. A multiple criteria model for the allocation of data files in a distributed information system. Computers Operation Research, vol. 19, No. 1, (1992), 21-33.

[17] Minsky, N. H. and Ungureanu, V. Unified Support for Heterogeneous Security Policies in Distributed Systems, Seventh USENIX Security Symposium, (1998)

[18] Lam, K., Pang, C., Son, S. H., and Cao, J., Resolving Executing-Committing Conflicts in Distributed Real-time Database Systems, The Computer Journal, Vol. 42, No. 8, (1999), 674-692

[19] Ram S., and Chastain C.L. Architecture of distributed data base systems. The Journal of Systems and Software, vol. 10, (1989), 77-95.

[20] Ram S., and Narasimhan S. Database Allocation in a distributed environment : Incorporating a concurrency control mechanism and queuing costs. Management Science, vol. 40, No. 8, (Aug. 94), 969-983

[21] Ramathirtham K., and Stankovic J.A. Scheduling algorithms and operating system support for real-time systems. Proceedings of IEEE, vol. 82, No. 1, (Jan. 1994), 55-66.

[22] Sha, L., Lehoczky, J. and Jensen, E. Modular concurrency control and failure recovery. IEEE Transactions on Computers, vol. 37, 146-159.

[23] Shin K., and Ramanathan P. Real-Time computing : A new discipline of computer science and engineering. Proceedings of IEEE, vol. 82, No. 1, (Jan. 1994), 6-23.

[24] Shu L., and Young M. Real-Time concurrency control with analytic worst case latency guarantees. Proceedings of 10th IEEE Workshop on Real-Time Operating Systems and Software, New York, (May 1993), 66-73.

[25] Sivasankaran, R. M., Stankovic, J. A., Towsley, D., Purimetla, B. and Ramamritham, K., Priority Assignment in Real-Time Active Databases, International Journal of Very Large Data Bases, Vol. 5, No. 1, January 1996, 19-34.

[26] Son, S. H., Beckinger, C. and Kim, Y. K.., MRDB: A Multi-user Real Time Database Testbed, Proceeding of 27th Hawaii International Conference on System Sciences, 1994, 543-552

[27] Son, S. H. and Park, S., Scheduling Transactions for Distributed Time-Critical Applications,, Readings in Distributed Computing Systems, edited by T. L. Casavant and M. Singhal, IEEE Computer Society Press, 1994

[28] Son, S. H., David, R. and Mukkamala, R, Supporting Security Requirements in Multilevel RealTime Databases, Proceedings of the IEEE Symposium on Security and Privacy, Oakland, CA, May 1995, 199-210

[29] Wolfson O. The overhead of locking (and commit) protocols in distributed databases. ACM Transactions on Database Systems, vol. 12, No. 3, (Sept. 1987), 453-471.

[30] Yu P.S., Wu K., Lin K., and Son S.H. On real-time databases : Concurrency Control and Scheduling. Proceedings of IEEE, vol. 82, No. 1, (Jan. 1994), 140-157.

[31] JAWS: Understanding High Performance Web Systems. Work in Progress, Object Technologies International and Eastman Kodak Company, http://www.cs.wustl.edu/~jxh/research/research.html

[32] Availability Analysis for Unisys e-@action Application Delivery System. http://www.unisys.com/hw/servers/enterprise/optimized/consolidation

ADDITIONAL INSTITUTE PAPERS AVAILABLE FROM THE KRANNERT GRADUATE SCHOOL OF MANAGEMENT

-1995-

1069   Sugato Chakravarty and John J. McConnell, AN ANAYLSIS OF PRICES, BID/ASK SPREADS, AND BID AND ASK DEPTH SURROUNDING IVAN BOESKY'S ILLEGAL TRADING IN CARNATION'S STOCK.

1070   John J. McConnell and Henri Servaes, EQUITY OWENERSHIP AND THE TWO FACES OF DEBT.

1071   Kenneth J. Matheny, REAL EFFECTS OF MONETARY POLICY IN A 'NEOCLASSICAL' MODEL: THE CASE OF INTEREST RATE TARGETING.

1072   Julie Hunsaker and Dan Kovenock, THE PATTERN OF EXIT FROM DECLINING INDUSTRIES.

1073   Kessan Joseph, Manohar U. Kalwani, THE IMPACT OF ENVIRONMENTAL UNCERTAINTY ON THE DESIGN OF SALESFORCE COMPENSATION PLANS.

1074   K. Tomak, A NOTE ON THE GOLDFELD QUANDT TEST

1075   Alok R. Chaturvedi, SIMDS: A SIMULATION ENVIRONMENT FOR THE DESIGN OF DISTRIBUTED DATABASE SYSTEMS

1076   Dan Kovenock and Suddhasatwa Roy, FREE RIDING IN NON-COOPERATIVE ENTRY DETERRENCE WITH DIFFERENTIATED PRODUCTS

1077   Kenneth Matheny, THE MACROECONOMICS OF SELF-FULFILLING PROPHECIES

1078   Paul Alsemgeest, Charles Noussair and Mark Olson, EXPERIMENTAL COMPARISONS OF AUCTIONS UNDER SINGLE-AND MULTI-UNIT DEMAND

1079   Dan Kovenock, Casper D de Vries, FIAT EXCHANGE IN FINITE ECONOMIES

1080   Dan Kovenock, Suddhasatwa Roy, DYNAMIC CAPACITY CHOICE IN A BERTRAND-EDGEWORTH FRAMEWORK

1081   Burak Kazaz, Canan Sepil, PROJECT SCHEDULING WITH DISCOUNTED CASH FLOWS AND PROGRESS PAYMENTS

-1996-

1082   Murat Koksalan, Oya Rizi, A VISUAL INTRACTIVE APPROACH FOR MULTIPLE CRITERIA DECISION MAKING WITH MONOTONE UTILITY FUNCTIONS

1083   Janet S. Netz, John D. Haveman, ALL IN THE FAMILY: FAMILY, INCOME, AND LABOR FORCE ATTACHMENT

APPENDICES FOR COLLABORATING TO COMPETE: A GAME-THEORETIC MODEL AND EXPERIMENTAL INVESTIGATION OF THE EFFECT OF PROFIT-SHARING ARRANGEMENT AND TYPE OF ALLIANCE ON RESOURCE-COMMITMENT DECISIONS

-2000-

1127   Sugato Chakravarty and Kai Li, AN ANALYSIS OF OWN ACCOUNT TRADING BY DUAL TRADERS IN FUTURES MARKETS: A BAYESIAN APPROACH

1128   Sugato Chakravarty, STEALTH TRADING: THE NEXT GENERATION

1129   S.G. Badrinath and Sugato Chakravarty, ARE ANALYST RECOMMENDATIONS INFORMATIVE?

1130   Sugato Chakravarty and Asani Sarkar, THE DETERMINANTS OF LIQUIDITY IN U.S. CORPORATE, MUNICIPAL AND TREASURY BOND MARKETS

1131   Vivian Lei and Charles Noussair, AN EXPERIMENTAL TEST OF AN OPTIMAL GROWTH MODEL

1132   Paul Healy and Charles Noussair, BIDDING BEHAVIOR IN THE PRICE IS RIGHT GAME: AN EXPERIMENTAL STUDY

1133   Kent D. Miller and Zur Shapira, BEHAVIORAL OPTION THEORY: FOUNDATIONS AND EVIDENCE

1134   Kent D. Miller, KNOWLEDGE INVENTORIES AND MANAGERIAL MYOPIA

1135   Gabriele Camera, Charles Noussair, and Steven Tucker, RATE-OF-RETURN DOMINANCE AND EFFICIENCY IN AN EXPERIMENTAL ECONOMY

1136   Timothy B. Folta, Jay J. Janney, SIGNALING FOR RESOURCE ACQUISITION: PRIVATE EQUITY PLACEMENTS BY TECHNOLOGY FIRMS

1137   Michael R. Baye, Dan Kovenock, Casper G. de Vries, COMPARATIVE ANALYSIS OF LITIGATION SYSTEMS: AN AUCTION-THEORETIC APPROACH

1138   Sugato Chakravarty, Asani Sarkar, DO DIFFERENCES IN TRANSPARENCY AFFECT TRADING COSTS? EVIDENCE FROM U.S. CORPORATE, MUNICIPAL AND TREASURY BOND MARKETS

1139   Charles Noussair, Stephane Robin, Bernard Ruffieux, GENETICALLY MODIFIED ORGANISMS IN THE FOOD SUPPLY: PUBLIC OPINION VS CONSUMER BEHAVIOR

1140   Gabriele Camera, SEARCH, DEALERS, AND THE TERMS OF TRADE