Appears in *Proc. Conf. Parallel and Distributed Computing Systems*, Dijon, France, 1996.

Performance Impact of I/O on Sender-Initiated and Receiver-Initiated Load Sharing Policies in Distributed Systems

Sivarama P. Dandamudi
School of Computer Science, Carleton University
Ottawa, Ontario K1S 5B6, Canada

Hamid Hadavi
Bell Northern Research
Ottawa, Ontario, Canada

TECHNICAL REPORT TR-96-23

ABSTRACT - Performance of distributed systems can be improved by distributing system workload from heavily loaded nodes to lightly loaded nodes (i.e., by load sharing). Previous studies have considered two dynamic load sharing policies: sender-initiated and receiver-initiated. In the sender-initiated policy, a heavily loaded node attempts to transfer work to a lightly loaded node and in the receiver-initiated policy a lightly loaded node attempts to get work from a heavily loaded node. Almost all the previous studies have neglected the I/O requirements of the jobs in evaluating the performance of various load sharing policies. In addition, sensitivity to variance in job inter-arrival and service times has not been studied. The goal of this paper is to fill the void in the existing literature. This paper models job's I/O requirements and studies the impact of job I/O service demand on the performance of sender-initiated and receiver-initiated policies. Furthermore, we also look at the impact of variance in inter-arrival times and in job service times (both processor and disk service time variances are considered).

Key words: Distributed systems, Load sharing, Performance evaluation, Receiver-initiated policy, Sender-initiated policy, Threshold policies.

1. INTRODUCTION

Load sharing is a technique to improve the performance of distributed systems by distributing the system workload from heavily loaded nodes, where service is poor, to lightly loaded nodes in the system. Load sharing policies may be either static or dynamic. Static policies use information only about the average system behaviour while the dynamic policies use the current or recent system state information. The chief advantage of the static policies is their simplicity (i.e., no need to collect system state information). The disadvantage of these policies is that they cannot respond to changes in system state (thus the performance improvement is limited). On the other hand, dynamic policies react to the system state changes dynamically and therefore provide substantial performance improvements over the static policies. However, this additional performance benefit is obtained at a cost - the cost of collecting and maintaining the system state information.

Two important components of a dynamic policy are a transfer policy and a location policy [4,5]. The transfer policy determines whether a job is processed locally or remotely and the location policy determines the node to which a job selected for possible remote execution should be sent. Typically, transfer policies use some kind of load index threshold to determine whether the node is heavily loaded or not. Several load indices such as the CPU queue length, time averaged CPU length, CPU utilization, the amount of available memory etc. have been proposed/used [18,21]. It has been reported that the choice of load index has considerable effect on the performance and that the simple CPU queue length load index was found to be the most effective [10].

The dynamic policies can employ either a centralized or a distributed location policy. In the centralized policy, state information is collected by a single node and other nodes would have to consult this node for advice on the system state. In the distributed policy, system state information is distributed to all nodes in the system. The centralized policy has the obvious disadvantages of diminished fault-tolerance and the potential for performance bottleneck (although some studies have shown that the node collecting and maintaining the system state need not be a performance bottleneck for reasonably large distributed systems [19]). The use of the centralized policy is often limited to a cluster of nodes in a large distributed system [22]. The distributed control eliminates these disadvantages associated with the centralized policy; however, distributed policy may cause performance problems as the state information may have to be transmitted to all nodes in the system. (Previous studies have shown that this overhead can be substantially reduced [4,5]).

Location policies can be divided into two basic classes: *sender-initiated* or *receiver-initiated*. In sender-initiated policies, congested nodes attempt to transfer work to lightly loaded nodes. In receiver-initiated policies, lightly loaded nodes search for congested nodes from which work may be transferred. It has been shown that, for CPU-intensive workloads and when the first-come/first-served (FCFS) scheduling policy is used, sender-initiated policies perform better than receiver-initiated policies at low to moderate system loads [5]. This is because, at these system loads, the probability of finding a lightly loaded node is higher than the probability of finding a heavily loaded node. At high system loads, on the other hand, receiver-initiated policies are better because it is much easier to find a heavily loaded node at these system loads. There have also been proposals to incorporate the good features of both these policies [17].

In the remainder of this section, we briefly review related work and outline the paper. A lot of literature has been published on this topic (an excellent overview of the topic can be found in [18]) [1-7,9-14,16-22]. However, most of the studies have concentrated on CPU-intensive jobs by neglecting the I/O requirements of jobs. Here, for the sake of brevity, we will mention a couple studies that have been done on CPU-intensive job workload. Eager et al. [4,5] provide an analytic study of dynamic load sharing policies. They showed that the sender-initiated location policy performs better at low to moderate system loads and the receiver-initiated policy performs better at high system loads. Further, they also showed that the overhead associated with state information collection and maintenance under the distributed policy can be reduced substantially (by probing only a few nodes about their system state as opposed to all nodes in the system). Shivaratri and Krueger [17] have proposed and evaluated, using simulation, two location policies that combine the good features of the sender-initiated and receiver-initiated location policies.

Hac and Jin [7] have implemented a receiver-initiated algorithm and evaluated its performance under three workload types: CPU-intensive, IO-intensive, and mixed workloads. They compare the performance of their load balancing policy with that when no load balancing is employed. They conclude that, for all the three types of workload, load balancing is beneficial. Unfortunately, they do not compare the performance of sender-initiated and receiver-

initiated policies. In addition, performance sensitivity to various system and workload parameters has not been studied.

A common thread among the previous studies (except for [7]) is that they are aimed at CPU-intensive jobs; as a consequence job's I/O requirements are completely neglected. Furthermore, analyses and simulations in these studies have been done under the assumption that the job service times are exponentially distributed and the the job arrivals form a Poisson process (i.e., job inter-arrival times are exponentially distributed). The goal of this paper is to fill the void in the existing literature. As a result, this paper concentrates on the impact of job I/O requirements on the performance of the sender-initiated and receiver-initiated dynamic load sharing policies. Furthermore, we also study the impact of variance in the inter-arrival times and in the job service times for CPU and disk.

The remainder of the paper is organized as follows. Section 2 discusses the two load sharing policies. Section 3 describes the workload and system models we have used in performing this study. The results are discussed in Section 4 and Section 5 concludes the paper by summarizing the results.

2. LOAD SHARING POLICIES

In our load sharing policies, as in most previous studies, only non-executing jobs are transferred for possible remote execution. The rationale is that it is expensive to migrate active jobs and, for performance improvement, such active job migration is strictly not necessary. It has been shown that, except in some extreme cases, active job migration does not yield any significant additional performance benefit [6,9]. Load sharing facilities like Utopia [22] will not consider migrating active jobs. As a result, we assume that each node has two queues that hold the jobs: a *job queue*, which is a queue of all the jobs (i.e., active jobs) that are to be executed locally and a *job waiting queue* that will hold those jobs that are waiting to be assigned to a node. Note that only jobs in this latter queue are eligible for load distribution.

We study two dynamic load sharing policies -- sender-initiated and receiver-initiated policies. In both policies, we use the same threshold-based transfer policy [4,5,17,18]. When a new job arrives at a node, the transfer policy looks at the job queue length at the node. It transfers the new job to the job queue (i.e., for local execution) if the job queue length is less than the threshold T. Otherwise, the job is placed in the job waiting queue for a possible remote execution. The location policy, when invoked, will actually perform the node assignment. The two location policies are described below.

Sender-initiated policy

When a new job arrives at a node, the transfer policy described above would decide whether to place the job in the job queue or in the job waiting queue. If the job is placed in the job waiting queue, the job is eligible for transfer and the location policy is invoked. The location policy probes (up to) a maximum of probe limit P_l randomly selected nodes to locate a node with the job queue length less than T. If such a node is found, the job is transferred for remote execution. The transferred job is directly placed in the destination node's job queue when it arrives. Note that probing stops as soon as a suitable target node is found. If all of the probes fail to locate a suitable node, the job is moved to the job queue to be processed locally. When a transferred job arrives at the destination node, the node must accept and process the transferred job even if the state of the node at that instance has changed since the probing.

Receiver-initiated policy

When a new job arrives at node S, the transfer policy would place the job either in the job queue or in the job waiting queue of node S as described above. The location policy is invoked by nodes only at times when a job leaves the CPU (either after receiving the required CPU service or to receive service from a disk). The location policy of node S attempts to transfer a job from its job waiting queue to its job queue if the job waiting queue is not empty. Otherwise, if the job queue length of node S is less than T, it initiates the probing process as in the sender-initiated policy to locate a node D with a non-empty job waiting queue. If such a node is found within P_l probes, a job from the job waiting queue of node D is transferred to the job queue of node S.

In this policy, if all of the probes fail to locate a suitable node to get work from, the node waits for the arrival of a local job. Thus, job transfers are initiated at most once every time a job leaves the CPU. This may cause performance problems because processing power is wasted until the arrival of a new job locally or return of a job after completing disk service. This poses performance problems if the load is not homogeneous (e.g. if only a few nodes are generating the system load) [18]. This adverse impact on performance can be remedied by, for example, reinitiating the load distribution activity after the elapse of a predetermined time if the node is still idle. Since we are considering a homogeneous system (all nodes are identical and load generation is identical at each node) we do not consider these modifications to the receiver-initiated policy.

3. SYSTEM AND WORKLOAD MODELS

3.1. System Model

In the simulation model, a locally distributed system is represented by a collection of N processor nodes and D disk nodes. In this paper we consider only homogeneous processor and disk nodes. We model the communication network in the system. The CPU would give preemptive priority to communication activities (such as reading a message received by the communication processor, initiating the communication processor to send a probe message etc.) over the processing of jobs.

The CPU overheads to send/receive a probe and to transfer a job are modelled by T_{probe} and T_{jx}, respectively. Actual job transmission (handled by the communication processor) time is assumed to be uniformly distributed between U_{jx} and L_{jx}. Probing is assumed to be done serially, not in parallel. For example, the implementation [3] uses serial probing. We do not, however, explicitly model the communication delays associated with disk service. Instead, communication delay is implicitly modelled by including it as a part of disk service time. Since disk service times dominate the communication delays, this assumption does not significantly affect our conclusions.

3.2. Workload Model

The system workload is represented by the following parameters. The job arrival process at each node is characterized by a mean inter-arrival time $1/\lambda$ and a coefficient of variation C_a. Jobs are characterized by a processor service demand (with mean $1/\mu$) and a coefficient of variation C_s. The I/O service demand of jobs is represented by a mean ($1/\mu_{io}$) with a coefficient of variation C_{io}. We study the performance sensitivity to variance in inter-arrival times as well as CPU and I/O service times (the CV values are varied up to 4). We use a two-stage hyperexponential model (see [8] for details) to generate service time and inter-arrival time CVs greater than 1 as in the previous studies.

A fraction α of the jobs arriving at each node is assumed to be IO-jobs (i.e., jobs with some disk service requirement) and the remaining $(1–\alpha)$ fraction of the jobs are CPU-intensive. Thus, by varying α value, we can model a purely CPU-intensive workload ($\alpha = 0$), purely IO workload ($\alpha = 1$), or mix of the two types of workload ($0 < \alpha < 1$). For the results reported here, we have used $\alpha = 0.5$ (i.e., half the jobs are CPU-intensive and the rest are IO-jobs). We, however, report the performance sensitivity to this parameter to understand the impact of this parameter.

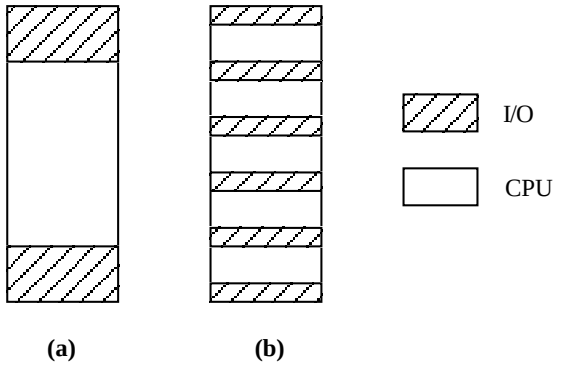

Figure 3.1 Two I/O models (a) Non-interleaved (b) interleaved

Given that the total I/O service demand of a job can be specified by μ_{io} and C_{io}, there is still the question of how this service demand is distributed within a job. We model I/O service in one two ways (as in [15]): non-interleaved I/O or interleaved I/O.

In the non-interleaved model (NI), two I/O operations are performed -- one at the beginning and the other at the end of job execution (see Figure 3.1a). The first I/O operation, for example, can represent reading input data and the second I/O operation can represent writing the results to disk. In this model, the total I/O service demand of a job is equally divided between the two operations. This type of I/O model is appropriate for applications that process files sequentially.

In the interleaved model (I), I/O service is interleaved with CPU service as shown in Figure 3.1b. In this model, the total I/O service demand is equally divided among n I/O operations. Similarly, the total CPU demand is equally distributed as shown in Figure 3.1b. This type model is suitable for applications that require, for example, random file access. The parameter n is an input parameter to the simulation model. In this study, we fix the value of n at 5 (due to space limitations, performance sensitivity to this parameter is not considered).

3.3. Performance Metrics

We use the mean response time as the chief performance metric to compare the performance of the various scheduling policies. In addition, we also obtain the probe rate per node and job transfer rate per node statistics from the simulation to explain the results. These two rates are computed as follows:

$$\text{Probe rate} = \frac{\text{number of probes during a simulation run}}{T * \text{number of nodes}}$$

$$\text{Transfer rate} = \frac{\text{number of job transfers during a simulation run}}{T * \text{number of nodes}}$$

where T is the simulation run length.

4. Performance Analysis

This section presents the simulation results and discusses the performance sensitivity of the sender-initiated and receiver-initiated load sharing policies. Unless otherwise specified, the following default parameter values are assumed. The distributed system has N = 32 CPU nodes interconnected by a 10 mega-bit/second communication network. The processor scheduling policy is first-come/first-served (FCFS). The number disk nodes D is fixed at 8. The average CPU service demand $1/\mu$ is one time unit (e.g., 1 second) and the disk service rate μ_{io} is 2.5. The percentage of IO-jobs α is fixed at 50%. The IO-jobs are either all non-interleaved (NI) or all interleaved (I). Since we are considering sender-initiated (SI) and receiver-initiated (RI) load sharing policies, each graph presented in this section will have four lines corresponding to SI-NI, SI-I, RI-NI, and RI-I. The size of a probe message is 16 bytes. The CPU overhead to send/receive a probe message T_{probe} is 0.003 time units and to transfer a job T_{jx} is 0.02 time units. Job transfer communication overhead is uniformly distributed between L_{jx} = 0.009 and U_{jx} = 0.011 time units (average job transfer communication overhead is 1% of the average job service time). Since we consider only non-executing jobs for transfer, 1% is a reasonable value. Note that transferring active jobs would incur substantial overhead as the system state would also have to be transferred. Threshold T for all policies is fixed at 2 (Section 4.7 discusses the impact of the threshold value). Probe limit P_l is 3 for all policies (Section 4.6 discusses the performance sensitivity to this parameter).

Batch strategy has been used to compute confidence intervals (30 batch runs, with each batch consisting of 10^5 to 10^6 jobs, were used for the results reported here). This strategy has produced 95% confidence intervals that were less than 2% of the mean response times when the system utilization is not very high.

4.1. Performance as a function of system load

Figure 4.1 shows the mean response time, probe and transfer rates for the two load sharing policies under the two I/O models. It can be observed from the data presented in this figure that the actual I/O model (non-interleaved or interleaved) does not significantly affect the performance of either of the two load sharing policies at low to moderate processor loads. At these processor loads, the sender-initiated policy performs better than the receiver-initiated policy for both non-interleaved and interleaved I/O models.

At very high processor loads (at about 90%), receiver-initiated policies move the system to the no load sharing case. This can be seen from Figures 4.1b and 4.1c where both the probe and transfer rates go to zero. This causes the response time to increase dramatically. The sender-initiated policies, on the other hand, still perform some load distribution even though the job transfer rate drops for processor loads higher than 0.8 (see Figure 4.1c). For the SI policies, the probe rate increases with system load as the probability of finding a node in the sender state (i.e., above threshold) increases with load. However, at high system loads, most of these probes are useless as there are fewer receiver nodes (i.e., nodes below the threshold level) in the system resulting in decreased job transfers.

In RI policies, the probe rate initially increases as the system load increases (in our case, up to about 60-70%). This is because load distribution activity is initiated every time a job leaves the CPU if that causes the node to fall below the threshold level. Thus, the frequency of this increases as the job arrival rate increases. The probe rate is substantially higher for the interleaved I/O model (see RI-I line in Figure 4.1b) than that for the non-interleaved I/O model. This is because a job with interleaved model departs the CPU for disk service much more frequently than a job with non-interleaved model. However, at low to moderate system loads, the probability of finding a sender node is low. Thus, the increased probing activity generated by RI-I is not useful unless the system load is high. Therefore, the interleaved model fosters increased load sharing and results in better performance at high system loads

4.2. Sensitivity to variance in inter-arrival times

This section considers the impact of inter-arrival time CV C_a on the performance of the sender-initiated and receiver-initiated policies. Figure 4.2 shows the mean response time, average probe rate and the job transfer rate when the processor load is 80% (i.e., λ = 0.8). The CPU and I/O service time CVs are fixed at 1. All other parameter values are set to their default values. The main observation from this figure is that the impact of the variance in inter-arrival times is much more significant on the receiver-initiated policies than on the sender-initiated policies, particularly when the non-interleaved I/O model is used.

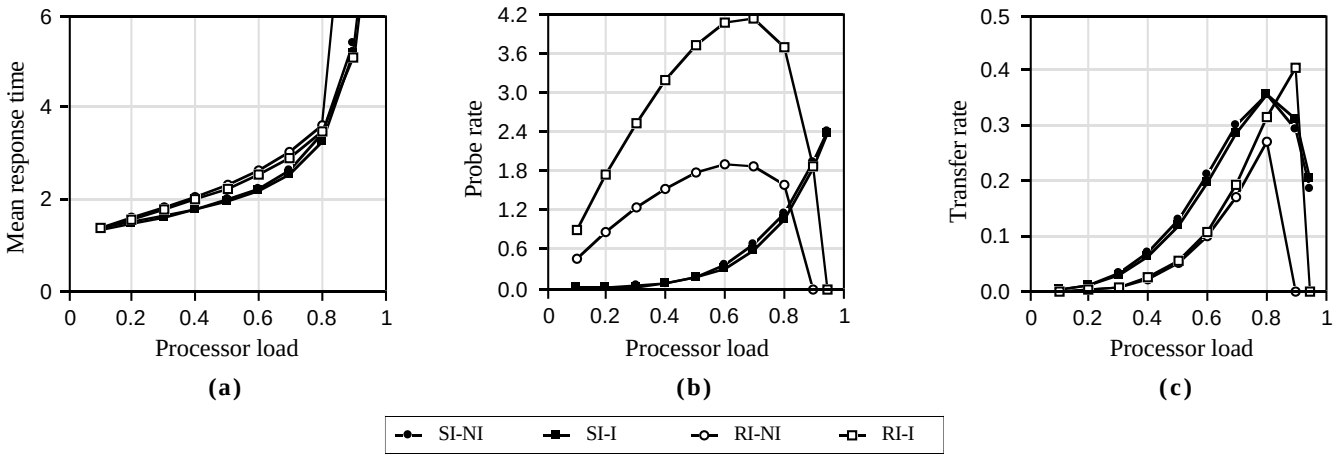

Figure 4.1 Performance sensitivity as a function system load
(λ is varied; $C_a = C_s = C_{io} = 1$; all other parameters are set to their default values)

The sender-initiated policies are relatively less sensitive to the inter-arrival CV. The receiver-initiated policies, on the other hand, exhibit increased sensitivity to C_a. The reason for this sensitivity of RI policies is that increased CV implies the clustered nature of job arrivals into the system; the higher the CV the more clustered the job arrival process is. This decreases the probability of finding a node in the receiver state (i.e., below the threshold level) and hence results in reduced probe rate and job transfer rate. Thus, increasing the inter-arrival CV moves the system based on the receiver-initiated policy (for non-interleaved model) towards no load sharing system. For example, assume that $C_s = 0$. If C_a is also zero, each node invokes the probing process to locate a sender node after the completion of each job (at the CPU) because local job arrival would be some time after the completion of the previous job (because of no variations either in service times or in inter-arrival times). Now increasing C_a means clustered arrival of jobs. Suppose 4 jobs have arrived into the system as a cluster. Then, the node would not initiate any load sharing activity until it completes these four jobs. Thus, on average, the probability of finding a node in the receiver state is a decreasing function of C_a (see Figure 4.2b). The RI-I substantially less sensitive to inter-arrival CV because it initiates load sharing activity much more frequently for reasons explained in Section 4.1. This is also demonstrated by Figures 4.2b and 4.2c. At higher C_a values, the probe rate of RI-NI drops well below that of the other three policies, thus moving the system toward no load sharing case. This causes substantial increase in response times (for example, at C_a =4).

On the other hand, clustered arrival of jobs fosters increased load sharing activity in sender-initiated policies. Thus the probe rate and the job transfer rate increase with increasing C_a (see Figures 4.2b and 4.2c). However, there is no significant difference between the SI-NI and SI-I policies for the C_a values considered here.

4.3. Sensitivity to variance in processor service times

This section considers the impact of service time CV on the performance of the sender-initiated and receiver-initiated policies. Figure 4.3 shows the mean response time, average probe rate and the job transfer rate when the processor load is 80% and the inter-arrival time and I/O CVs are fixed at 1. All other parameter values are set to their default values. In this context, it should be noted that the service time distribution of the data collected by Leland and Ott [11] from over 9.5 million processes has been shown to have a coefficient of variation of 5.3 [9].

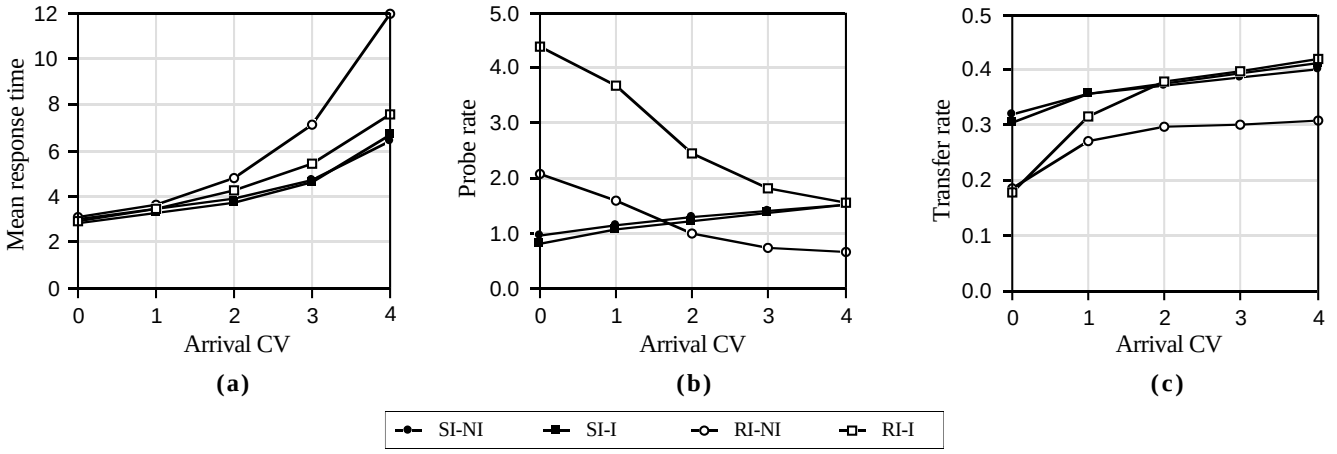

Figure 4.2 Performance sensitivity to variance in job inter-arrival times
($\lambda = 0.8$, C_a is varied; $C_s = C_{io} = 1$; all other parameters are set to their default values)

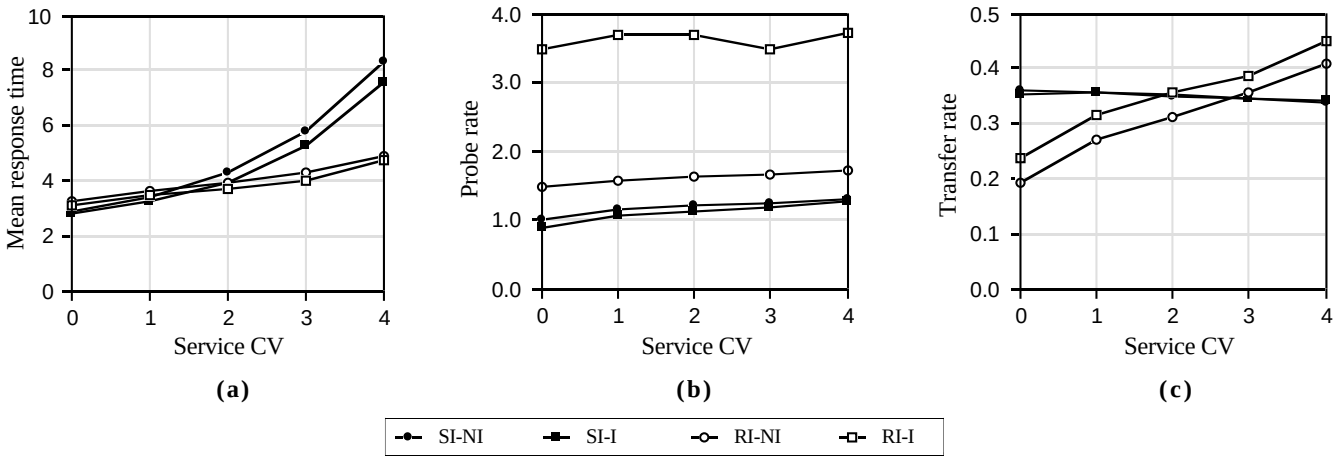

Figure 4.3 Performance sensitivity to variance in processor service demand
(λ = 0.8, C_s is varied; $C_a = C_{io}$ = 1; all other parameters are set to their default values)

It can be observed from Figure 4.3 that the sender-initiated policies exhibit more sensitivity to C_s than the receiver-initiated policies. At high Cs (for example, when Cs = 4) receiver-initiated policies perform substantially better than the sender-initiated policies as shown in Figure 4.3a. The sensitivity of the FCFS scheduling policy to variance in service times is an expected result because larger jobs can monopolize the processing power that results in increased probe rates (Figure 4.3b). However, this increase in probing activity is useless because, at high system loads, the probability of finding a receiver node is small (see Figure 4.3c).

At high C_s, the receiver-initiated policies perform substantially better than the sender-initiated policies mainly because, at high system loads, the probability of finding an overloaded node is high. Thus, in this case, increased probing activity results in increased job transfers (thereby increasing load sharing). This results in a better performance and reduced sensitivity to C_s. The RI-I performs better than RI-NI because of the higher frequency at which the load sharing activity is initiated as the jobs leave the CPU several times for disk service. As can be seen from Figure 4.3c, RI-I is successful in transferring more jobs than RI-I policy for the CPU service time CVs shown in Figure 4.3.

4.4. Sensitivity to variance in disk service times

This section considers the impact of variance in disk service times. As in the last two sections, the other two CVs are set at 1. Figure 4.4 shows the mean response time, average probe rate and the job transfer rate when the processor load is 80%. It can be observed from this figure that C_{io} affects all four policies similarly and the difference in response times is not substantial. Sender-initiated policies perform marginally better than their receiver-initiated counterparts.

For the receiver-initiated policies, the probe rate decreases with increasing variance disk service times. This is because as C_{io} increases, the jobs depart the disks in a clustered fashion causing the jobs to return to CPU for service in a clustered manner. This has the same effect as the inter-arrival CV (see Section 4.2). This effect is more pronounced for the interleaved I/O model. As shown in Figure 4.4b, the decrease in probe rate for RI-I is more than that for RI-NI policy. For reasons explained in Section 4.2, the sender-initiated policies see an increased probe rate with disk service time variance (though only marginal in this case).

Note that the results presented in Figure 4.4 were obtained by setting job arrival rate to 0.8. At this arrival rate the average disk utilization is about 64%. The response time sensitivity to disk service time variance increases with increasing disk utilization.

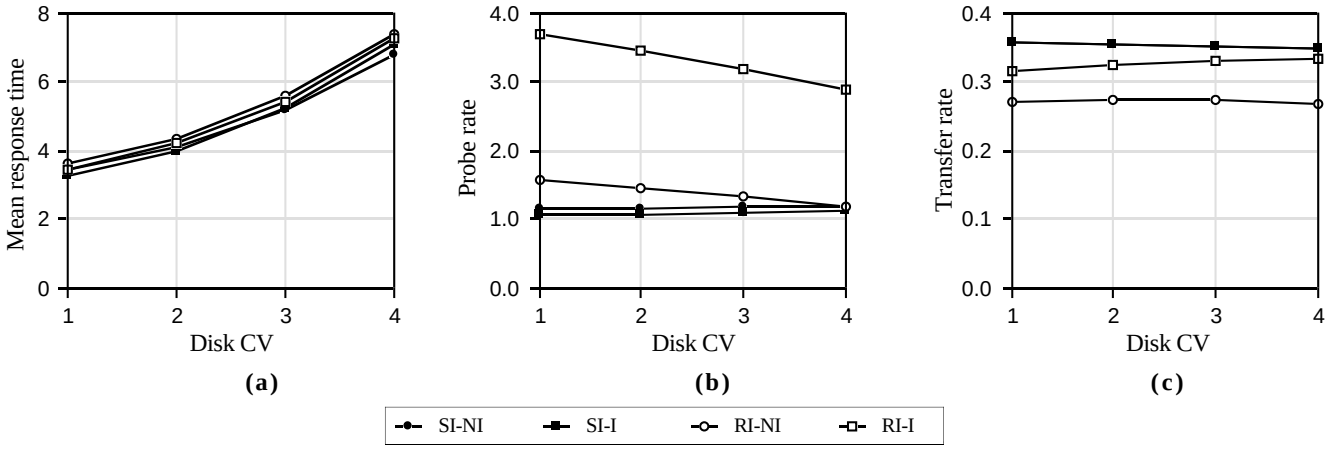

Figure 4.4 Performance sensitivity to variance in disk service demand
(λ = 0.8, C_{io} is varied; $C_a = C_s$ = 1; all other parameters are set to their default values) SI-NI and SI-I lines overlap in figure (c).

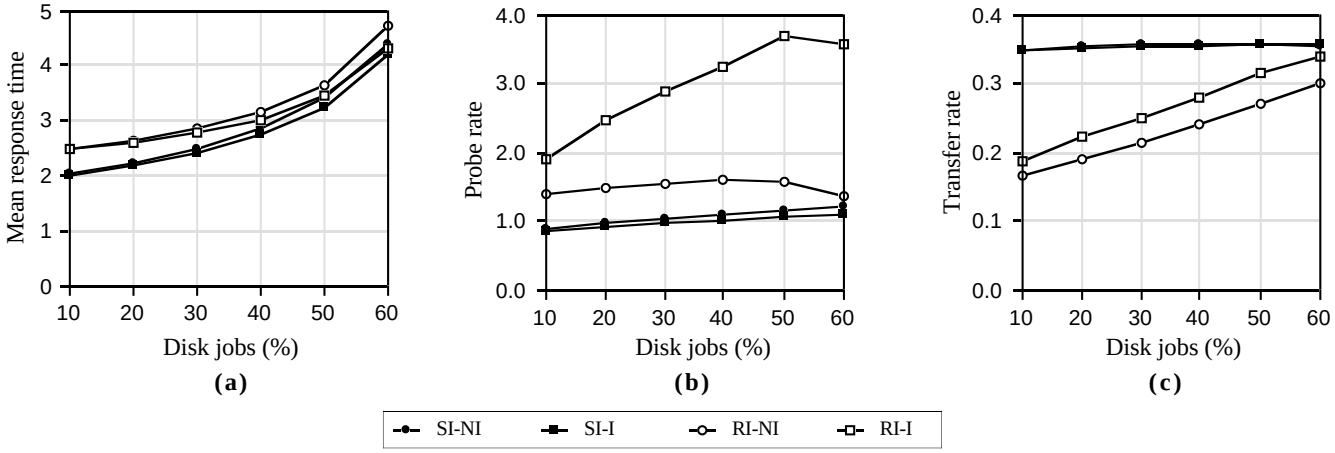

Figure 4.5 Performance sensitivity to percentage of jobs requiring disk access
(λ = 0.8, fraction of disk jobs α is varied; $C_a = C_s = C_{io} = 1$; all other parameters are set to their default values)

4.5. Performance sensitivity to disk jobs

The data presented in previous sections assumed an equal proportion of CPU-intensive jobs and IO-jobs (i.e., α = 50%). The effect of this parameter is studied in this section. The results are presented in Figure 4.5. As in previous sections, the processor load is 80% and all three CVs are fixed at 1. We can make two observations form the data presented in this figure.

(i) The difference in response time between interleaved and non-interleaved models for the same load sharing policy increases with the proportion of disk jobs. For example, the difference between the response times of RI-I and RI-NI increases as the percentage of disk jobs increases. For example, when α=10%, both RI-I and RI-NI provide the same response times. When α=60%, the difference is about 10%. Similar observations can be made for SI-I and SI-NI policies as well.

(ii) The receiver-initiated policies improve their performance with increasing proportion of disk jobs. As shown in Figure 4.5c, the job transfer rate of receiver-initiated policies increases with the proportion of disk jobs. On the other hand, there is no significant change for the sender-initiated policies.

4.6. Performance sensitivity to probe limit

The previous sections assumed a probe limit of 3. This section discusses the impact of the probe limit on the performance of the sender-initiated and receiver-initiated policies. Figure 4.6 shows the mean response time, average probe rate and the job transfer rate when the processor load is 80%. The processor and I/O service time CVs and the inter-arrival time CV are fixed at 1. All other parameter values are set to their default values.

Receiver-initiated policies

The response times of RI-NI and RI-I policies tend to converge as the probe limit increases. For the data presented in Figure 4.6a, both policies provide similar response times when the probe limit is greater then or equal to 6. It can be observed from this figure that the RI-NI experiences more improvement in response time than RI-I policy. As shown in Figure 4.6b, the probe rate of RI-NI policy increases with the probe limit whereas the probe rate of RI-I is flat beyond the probe limit of 6. This translates to reduced job transfer rate as shown in Figure 4.6c. The reason for the flat probe rate in RI-I policy is that the probability of a job arriving for CPU service after finishing the disk service increases with increasing probe limit (as we use serial probing). This prob-

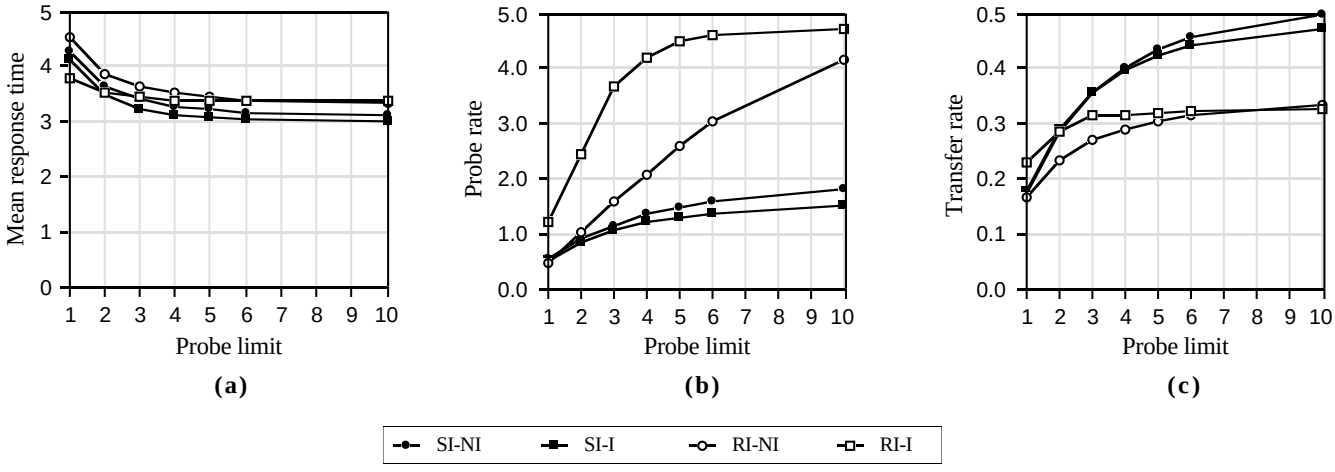

Figure 4.6 Performance sensitivity to probe limit
(λ = 0.8, probe limit P_l is varied ; $C_a = C_s = C_{io} = 1$; all other parameters are set to their default values)

ability is low with non-interleaved disk service. Notice that the transfer rate itself does not change as much as the probe rate even though the RI-NI exhibits a higher rate of change with probe limit. The main reason is that, at high processor loads considered, the probability of successfully finding a sender node is high with initial few probes. Therefore, further increases in probe limit do not result in substantial increase in job transfer rates.

Sender-initiated policies

In sender-initiated policies, probe rate change is not as high as that in the receiver-initiated policies (see Figure 4.6b). However, job transfer rates show a great degree of sensitivity to the probe limit as shown in Figure 4.6c. The reason is that, at high system loads there are likely more sender than receiver nodes, the probability of finding a receiver node increases with the probe limit. For higher probe limit values, the sender-initiated policies perform better than the receiver-initiated policies. Furthermore, the data in Figure 4.6a suggests that the SI-I maintains its performance superiority over SI-NI over the range of probe limit values considered here.

4.7. Performance sensitivity to threshold

The previous sections assumed a threshold of $T = 2$. This section discusses the impact of threshold on the performance of the sender-initiated and receiver-initiated policies. Figure 4.7 shows the mean response time, average probe rate and the job transfer rate when the processor load is 80%. The CPU and I/O service time CVs and the inter-arrival time CV are set to 1. All other parameter values are set to their default values.

Sender-initiated policies

In general, increasing the threshold value decreases the scope of load sharing. This manifests in decreased probe rates as the threshold is increased. It should be noted that $T = \infty$ corresponds to a system with no load sharing. At $T = 1$, all job arrivals, when the system is busy, would result in initiating load distribution activity but the probability of finding an idle node is very low. This, therefore, results in poor performance at high system loads as shown in Figure 4.7.

The two sender-initiated policies show performance improvements initially and further increase in threshold would result in performance deterioration as shown in Figure 4.7a. The reason for the initial performance improvement is that increasing the threshold from 1 increases the probability of finding a node that is below the threshold and therefore results in increased job transfer rate even though the probe rate is reduced (Figures 4.7b and 4.7c). Further increase in the threshold value result in decreasing number of attempts at load distribution. This causes reduced probe rates and job transfer rates.

Receiver-initiated policies

The data presented in Figure 4.7a suggests that $T = 1$ is the best value. That is, load distribution should be attempted only when a node is idle (Livny and Melman [13] call it 'poll when idle' policy). The reason is that a node is better off avoiding load distribution when there is work to do. In general, the performance of the receiver-initiated policies deteriorates with increasing threshold value. The main reason for this is that the system moves towards diminished load sharing system. Note that even though the probe rate increases with the threshold value (Figure 4.7b), this does not result in finding a node that is above threshold. Thus, the transfer rate decreases with the threshold value (Figure 4.7c). The RI-I policy exhibits more sensitivity to the threshold parameter than the RI-NI policy. For example, at $T = 6$, the response time of RI-NI is better than that of RI-I policy.

5. Conclusions

Load sharing has been extensively studied by several researchers. There are two basic dynamic load sharing policies: sender-initiated and receiver-initiated. In sender-initiated policies, heavily loaded nodes attempt to transfer work to lightly loaded nodes and in receiver-initiated policies lightly loaded nodes attempt to get work from heavily loaded nodes. A common thread among the previous studies is that they have assumed only CPU-intensive jobs by completely ignoring their I/O requirements. In addition, sensitivity to variance in job inter-arrival and service times has not been studied. The goal of this paper has been to fill the void in the existing literature. As a result this paper has considered the impact job's I/O service demand on the performance of the sender-initiated and receiver-initiated dynamic load sharing policies. Furthermore, we have also studied the impact of variance in the inter-arrival times and in job CPU and disk service times.

We have used two I/O models: non-interleaved and interleaved models. In the non-interleaved (NI) model, I/O operations are done once at the beginning and once more at the end of a job. In the interleaved (I) model, I/O processing and CPU processing are interleaved. In comparing the performance of the sender-initiated and receiver-initiated policies using these two I/O models, we have shown that:

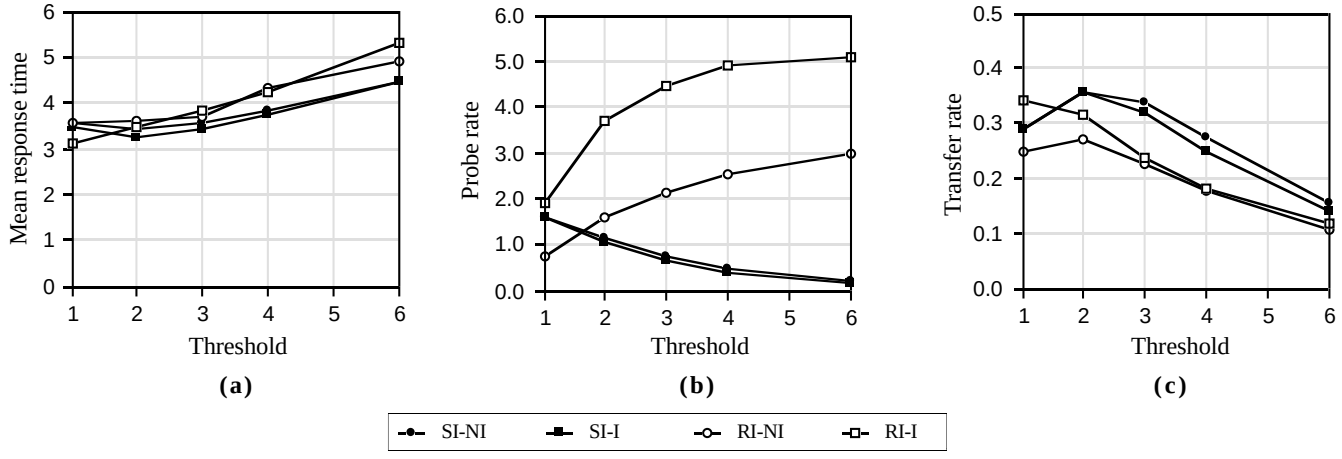

Figure 4.7 Performance sensitivity to the threshold parameter
($\lambda = 0.8$, threshold is varied ; $C_a = C_s = C_{io} = 1$; all other parameters are set to their default values)

- Exact I/O model has a more significant impact on receiver-initiated policies when the inter-arrival time variance is high (i.e., clustered arrival of jobs). The non-interleaved I/O model exhibits more sensitivity to this parameter.
- Sender-initiated policies are relatively more sensitive to variance in CPU service times. Again, the non-interleaved I/O model exhibits more sensitivity to this parameter.
- Variance in disk service times impacts both sender-initiated and receiver-initiated policies similarly.
- The performance difference between interleaved and non-interleaved I/O increases with the proportion of jobs with I/O requirements.

In this study, we have not considered the impact of system and workload heterogeneity [14,16]. System heterogeneity refers to non-homogeneous nodes (nodes with different processing speeds, for example) and the workload heterogeneity refers to non-homogeneous job arrival characteristics. In addition, the effect of various processor and disk scheduling policies needs to be studied. We intend to extend our work to study these issues in the near future.

ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada and Carleton University.

REFERENCES

[1] S. P. Dandamudi, "Performance Impact of Scheduling Discipline on Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE International Conference on Distributed Computing Systems*, Vancouver, May 1995, pp. 484-492.

[2] S. P. Dandamudi and M. Lo, *Hierarchical Load Sharing Policies for Distributed Systems*, Technical Report SCS-96-1, School of Computer Science, Carleton University, Ottawa, Canada (This technical report can be obtained from http://www.scs.carleton.ca)

[3] P. Dikshit, S. K. Tripathi, and P. Jalote, "SAHAYOG: A Test Bed for Evaluating Dynamic Load-Sharing Policies," Software - Practice and Experience, Vol. 19, No. 5, May 1989, pp. 411-435.

[4] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. Software Engng.*, Vol. SE-12, No. 5, May 1986, pp. 662-675.

[5] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing," *Performance Evaluation*, Vol. 6, March 1986, pp. 53-68.

[6] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "The Limited Performance Benefits of Migrating Active Processes for Load Sharing," *ACM Sigmetrics Conf.*, 1988, pp. 63-72.

[7] A. Hac and X. Jin, "Dynamic Load Balancing in a Distributed System Using a Decentralized Algorithm," *IEEE Int. Conf. Dist. Computing Systems*, 1987, pp. 170-177.

[8] H. Kobayashi, Modeling and Analysis: An Introduction to System Performance Evaluation Methodology, Addison-Wesley, 1981.

[9] P. Krueger and M. Livny, "A Comparison of Preemptive and Non-Preemptive Load Distributing," *IEEE Int. Conf. Dist. Computing Systems*, 1988, pp. 123-130.

[10] T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme," *IEEE Trans. Software Engng.*, Vol. SE-17, No. 7, July 1991, pp. 725-730.

[11] W. E. Leland and T. J. Ott, "Load Balancing Heuristics and Process Behavior," *ACM SIGMETRICS 86*, 1986, pp. 54-69.

[12] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor - A Hunter of Idle Workstations," *IEEE Int. Conf. Dist. Computing Systems*, 1988, pp. 104-111.

[13] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems, *Proc. ACM Computer Network Performance Symp.*, 1982, pp. 47-55.

[14] M. Lo and S. Dandamudi, "Performance of Hierarchical Load Sharing in Heterogeneous Distributed Systems," *Int. Conf. Parallel and Distributed Computing Systems*, Dijon, France, 1996. (this proceedings)

[15] S. Majumdar and Y. M. Leung, "Characterization of Applications wit I/O for Processor Scheduling in Multiprogrammed Parallel Systems," *Proc. IEEE Symp. Parallel Dist. Processing*, Dallas, 1994, pp. 298-307.

[16] R. Mirchandaney, D, Towsley, and J. A. Stankovic, "Adaptive Load Sharing in Heterogeneous Distributed Systems," *J. Parallel and Distributed Computing*, Vol. 9, 1990, pp. 331-346.

[17] N. G. Shivaratri and P. Krueger, "Two Adaptive Location Policies for Global Scheduling Algorithms," *IEEE Int. Conf. Dist. Computing Systems*, 1990, pp. 502-509.

[18] N. G. Shivaratri, P. Krueger, and M. Singhal, "Load Distributing for Locally Distributed Systems," *IEEE Computer*, December 1992, pp. 33-44.

[19] M.M. Theimer and K. A. Lantz, "Finding Idle Machines in a Workstation-Based Distributed System," *IEEE Int. Conf. Dist. Computing Systems*, 1988, pp. 112-122.

[20] Y. T. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems," *IEEE Trans, Computers*, March 1985, pp. 204-217.

[21] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *IEEE Trans. Software Engng.*, Vol. SE-14, No. 9, September 1988, pp. 1327-1341.

[22] S. Zhou, X. Zheng, J. Wang, and P. Delisle, "Utopia: a Load Sharing Facility for Large, Heterogeneous Distributed Computer Systems," *Software - Practice and Experience*, Vol. 23, No. 12, December 1993, pp. 1305-1336.