Appears in *Proc. MASCOTS'97, Haifa, Israel, January 1997.*

The Effect of Scheduling Discipline on Dynamic Load Sharing in Heterogeneous Distributed Systems

Sivarama P. Dandamudi
School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada
sivarama@scs.carleton.ca

TECHNICAL REPORT TR-96-26

ABSTRACT − Dynamic load sharing policies have been extensively studied. Most of the previous studies have assumed a homogeneous distributed system with a first-come-first-served (FCFS) node scheduling policy. In addition, job service times and inter-arrival times are assumed to be exponentially distributed. In this paper, we study the impact of these assumptions on the performance of *sender-initiated* and *receiver-initiated* dynamic load sharing policies in *heterogeneous* distributed systems. We consider two node scheduling policies − first-come/first-served (FCFS) and round robin (RR) policies. Furthermore, the impact of variance in inter-arrival times and job service times is studied. Our results show that, even in heterogeneous distributed systems, when the round robin node scheduling policy is used, sender-initiated policy is better than the receiver-initiated policy unless the variance in job service times is low. This is an important observation as most workstations use a scheduling policy similar to the round robin policy considered here.

1. Introduction

Load sharing improves performance of distributed systems by distributing system load from a heavily loaded node to a lightly loaded node. Dynamic load sharing policies use current (or most recent) system state information to perform load distribution. Two important components of a dynamic policy are a transfer policy and a location policy [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.

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, 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 [6]. This is because, at these system loads, the probability of finding a lightly loaded node is higher than that 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.

In the remainder of this section, we briefly review related work and outline the paper. Extensive literature exists on this topic [1-22]. For the sake of brevity, we will only review a subset of this literature that is directly relevant to our study. A common thread among the previous studies, with a few exceptions, is the assumption that the system is homogeneous and the node scheduling policy is first-come/first-served (FCFS). In addition, analyses and simulations in these studies have been done under the assumption that job service times as well as job inter-arrival times are exponentially distributed. We have reported the impact of scheduling discipline on the performance of dynamic load sharing policies in homogeneous systems [1]. The focus of this paper is on heterogeneous systems.

Mirchandani et al. [15] consider the impact of delay on the performance of heterogeneous distributed systems. They consider two types of heterogeneous systems: in type I systems external job arrival rates at nodes may differ and in type 2 systems the processing rates of the nodes may also differ. They use analytical models to estimate the performance. In order to facilitate analytical modeling, they assume that inter-arrival times and service times are exponentially distributed. The node scheduling policy is the non-preemptive first-come/first-served. This work is directly relevant to our study. As in their study, we consider the two types of heterogeneous systems. However, we focus on some of the gaps left by their study. We concentrate on non-exponential inter-arrival and service times, and use preemptive round robin node scheduling policy. Because of this focus, we have to use a simulation model for performance evaluation.

The remainder of the paper is organized as follows. Section 2 presents the load sharing and node scheduling policies. Section 3 describes the workload and system models used in this study. The results for type I and type II heterogeneous systems are discussed in Sections 4 and 5, respectively. Conclusions are given in Section 6.

2. Load Sharing and Node Scheduling Policies

This section presents the load sharing policies and the node scheduling policies. Section 2.1 describes the sender-initiated and receiver-initiated load sharing policies. The two node scheduling policies are described in Section 2.2.

2.1. Load Sharing Policies

Our load sharing policies are slightly different from those reported in the literature. These subtle changes are motivated by the following aspects of locally distributed systems. First, workstations use some kind of round robin scheduling policy for efficiency and performance reasons. Thus, it is necessary to have a number of jobs that is related to the degree of multi-programming (T_m). In the implementation of Hac and Jin [8], they have used a similar parameter (they call it capacity C). Second, it is expensive to migrate active processes and, for performance improvement, such active process migration is strictly not necessary. It has been shown that, except in some extreme cases, active process migration does not yield any significant additional performance benefit [7]. Load sharing facilities like Utopia [22] will not consider migrating active processes. 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 the job waiting queue are eligible for load distribution.

In both sender-initiated and receiver-initiated policies, we use the same threshold-based transfer policy. 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 degree of multiprogramming T_m. Otherwise, the job is placed in the job waiting queue of the node 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_s. 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 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 probing. In this paper, we assume that $T_m = T_s$ for the sender-initiated load sharing policy.

Receiver-initiated policy: When a new job arrives at node *R*, the transfer policy would place the job either in the job queue or in the job waiting queue of node *R* as described before. The location policy is invoked by nodes only at times of job completion. The location policy of node *R* 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 *R* is less than T_R, it initiates the probing process as in the sender-initiated policy to locate a node *S* 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 *S* will be transferred to the job queue of node *R*. In this paper, as in the previous literature [6,18], we assume that $T_R = 1$. That is, load distribution is attempted only when a node is idle. The motivation is that a node is better off avoiding load distribution when there is work to do. Furthermore, several studies have shown that a large percentage (up to 80% depending on time of day) of workstations are idle [10,13,16,19]. Thus the probability of finding an idle workstation is high.

In previous studies, if all probes failed 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 is completed. This may cause performance problems because the processing power is wasted until the arrival of a new job locally. This poses severe performance problems if the load is not homogeneous (e.g., if only a few nodes are generating the system load) [18] or when the variance in inter-arrival times is high [1]. In order to remedy this situation, the receiver initiated policy implemented here reinitiates load sharing activity after the lapse of P_I time since the last probe if the node is still idle. Such a strategy has been suggested but not implemented in previous studies [18].

2.2. Scheduling Policies

We consider two scheduling policies - one preemptive and one non-preemptive.

- *First-Come-First-Serve (FCFS):* This is a non-preemptive scheduling policy that schedules jobs in the order of their arrival into the job queue. This node scheduling policy has been used in most of the previous studies.
- *Round Robin (RR):* This is a preemptive scheduling policy. In this policy, each scheduled job is given a quantum *Q* and, if the job is not completed within that quantum, it is preempted and placed at the end of the associated job queue. This would eliminate a particularly large service demand job from monopolizing the processor. Note that each preemption would cause a context switch overhead *CSO*.

3. System and Workload Models

In the simulation model, a locally distributed system is represented by a collection of nodes. We model the communication network in the system at a higher level. We model communication delays without modelling the low-level protocol details. Each node is assumed to have a communication processor that is responsible for handling communication with other nodes. The CPU would give preemptive priority to communication activities (such as sending a probe message) over 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 [4] uses serial probing.

The system workload is represented by four parameters. The job arrival process at each class *i* node is characterized by a mean inter-arrival time $1/\lambda_i$ and a coefficient of variation C_{ai}. Jobs are characterized by a processor service demand on a class *i* node (with mean $1/\mu_i$) and a coefficient of variation C_{si}. We study the performance sensitivity to variance in both inter-arrival times and service times (the CV values are varied from 0 to 4).We use a two-stage hyperexponential model to generate service times and interarrival times with CVs greater than 1.

We consider two types of heterogeneous systems. In type I systems, the nodes are homogeneous (i.e., have the same processing rate) but the job arrival rates at nodes are different. To reduce the complexity of the experiments, we consider two node classes as in [15] and refer to them as class 1 and class 2. In type II systems, the node processing rates are also different. We consider two node classes in type II systems as well. Nodes in each class in a type II system can have different job arrival rates but all nodes in a given class have the same processing rate. However, processing rates of nodes are different for the two classes.

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

$$\text{Probe rate/node} = \frac{\text{\# of probes during simulation}}{(T * \text{number of nodes})}$$

$$\text{Transfer rate/node} = \frac{\text{\# of job transfers during simulation}}{(T * \text{number of nodes})}$$

where T is the simulation run length. While probe and transfer rates are used in our discussion to explain the behaviour of the system, the actual data on these two metrics are not presented due to space limitations.

4. Performance of Type I Systems

This section presents the simulation results for type I heterogeneous systems. Unless otherwise stated, the following default parameter values are assumed. The distributed system has $N = 32$ nodes interconnected by a 10 megabit/second communication network. The number of class 1 nodes N_1 is 6 and the number of class 2 nodes N_2 is 26. Section 4.2 discusses the impact of node distribution between class 1 and class 2. The average job service time is one time unit for both classes. That is $\mu_1 = \mu_2 = 1$. 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 (i.e., 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. Transfer policy threshold T_m is 2 and location policy threshold T_s (for the sender-initiated policies) is 2 and T_R (for the receiver-initiated policies) is 1 (as explained in Section 2.1). Similar threshold values are used in previous studies [4,5,6]. Section 4.6 discusses the impact of this parameter. Probe limit P_l is 3 for all policies. Section 4.5 considers the impact of the probe limit. For the RR policies, quantum size Q is 0.1 time unit and the context switch overhead CSO is 1% of the quantum size Q.

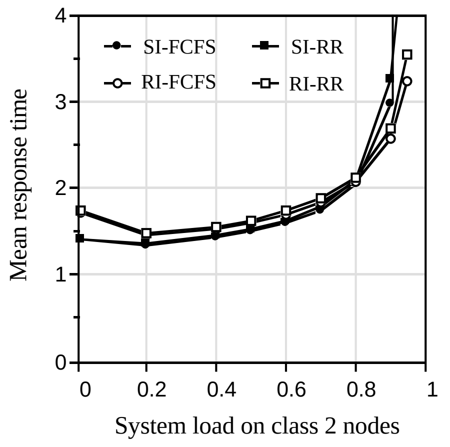
A batch strategy has been used to compute confidence intervals (at least 30 batch runs were used for the results reported here). This strategy produced 95% confidence intervals that were less than 1% of the mean response times when the system utilization is low to moderate and less than 5% for moderate to high system utilization (in most cases, up to about 90%).

4.1. Performance as a function of system load

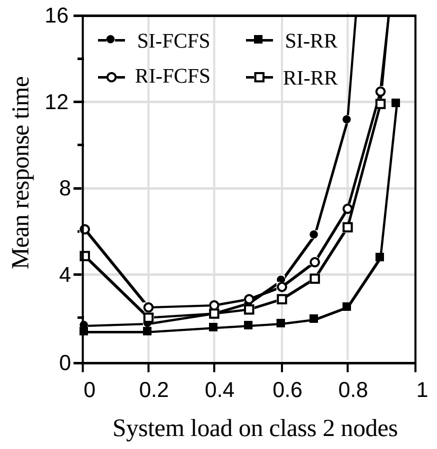
Figure 4.1 shows the mean response time as a function of offered system load. Note that the offered system load for a class i node is given by λ_i/μ_i. Since $\mu_1 = \mu_2 = 1$, offered system load on class i nodes is equal to λ_i. In this experiment we have fixed λ_1 at 0.9 and λ_2 is varied to see the impact of system load on performance.

The results in Figure 4.1a correspond to an inter-arrival CV (C_{ai}) and service time CV (C_{si}) of 1 (for both classes) and those in Figure 4.1b were obtained with the inter-arrival time and service time CVs of 4. As can be expected, higher service and inter-arrival CVs have substantial impact on all policies. When $C_{ai} = C_{si} = 1$ for both classes, scheduling policy has little impact except at very high system loads. At these system loads, FCFS performs better than RR because the service time CV is low. Also, receiver-initiated policies perform better than sender-initiated policies at high system loads (for class 2 load > 0.8).

When $C_{ai} = C_{Si} = 4$ (for both classes), sender-initiated policies perform better at low system loads. At low system load on class 2 nodes, the performance of the receiver-initiated policies deteriorates rapidly. The reason is that there are only 6 (out of 32) nodes operating at an offered system load of 0.9 while the remaining 26 nodes are operating at 0.01 system load. Since load sharing activity is initiated at most at $1/P_I = 1$ rate when the nodes are idle, the 6 heavily loaded nodes do not get much help from the remaining 26 nodes. The impact of this is more pronounced in Figure 4.1b because of the clustered arrival of jobs and the higher variance in service times.



(a) Inter-arrival CV C_{ai} = Service CV C_{si} = 1 for both classes



(a) Inter-arrival CV C_{ai} = Service CV C_{si} = 4 for both classes

Figure 4.1 Performance sensitivity as a function of offered system load (N = 32 nodes, $N_1 = 6$, $N_2 = 26$, $\mu_1 = \mu_2 = 1$, $\lambda_1 = 0.9$ and λ_2 is varied to vary system load on class 2 nodes, $T_m = 2$, $T_S = 2$, $T_R = 1$, $P_l = 3$, $P_I = 1$, transfer cost = 1%, $Q = 0.1$, CSO = 1% of Q)

At moderate to high system loads on class 2 nodes, SI-RR gives the best performance while SI-FCFS the worst. Performance of the two receiver-initiated policies falls in between that of SI-RR and SI-FCFS policies. The node scheduling policy has substantial impact on the sender-initiated policy. The worst performance of the FCFS policy is due to the sensitivity of the FCFS policy to service time CV (further discussed in Section 4.4). The RI-FCFS performs better because, at high system loads, sender-initiated policies would have to work harder to find a receiver node; however, for receiver-initiated policies, the probability of finding a sender node is high. Thus, even though the probe rate is high for the sender-initiated policies, job transfer rate is lower than that of the receiver-initiated policy. (Due to space limitation, probe and transfer rates are not presented).

The RI-RR performs marginally better than RI-FCFS while SI-RR provides substantially better performance. In fact, SI-RR exhibits the least sensitivity to system load among the four policies. There are several factors contributing to this performance superiority: (i) RR policy gives priority to smaller jobs through preemption by not allowing longer jobs to dominate the processors. (ii) Compared to FCFS, RR tries to finish smaller jobs faster; therefore, the probability of finding a node in receiver state increases with RR, which also implies that the probability of finding a node in the sender state decreases. These factors affect sender-initiated and receiver-initiated policies differently. In sender-initiated policy, clustered arrival of jobs causes more load sharing activity whereas in receiver-initiated policies, it causes the opposite effect. Since RR node scheduling policy increases the probability of finding a receiver node, it improves performance of the SI-RR policy. On the other hand, RR policy does not have as much influence on the performance of the receiver-initiated policy because the probability of finding a sender node decreases with RR node scheduling policy. Although it is not shown here, probe rate of RR policies is always lower than the corresponding values for the FCFS policy.

4.2. Impact of degree of heterogeneity

In this experiment, the offered system loads for class 1 and 2 are fixed as 0.1 and 0.9 (i.e., $\lambda_1 = 0.1$ and $\lambda_2 = 0.9$). Figure 4.2 shows the impact of degree of heterogeneity. In this graph, the x-axis gives the number of class 2 nodes N_2 and remaining nodes belong to class 1 group. Since class 1 nodes are lightly loaded, the mean response time increases with N_2.

FCFS Policy: When the number of class 2 nodes N_2 is smaller, SI-FCFS performs better than RI-FCFS because class 1 nodes are lightly loaded and the probability of finding a receiver is high in this case. As N_2 increases, the probability of finding a sender increases. As a result, the RI-FCFS performs better and the performance of SI-FCFS deteriorates rapidly at high N_2 values. In fact, the transfer rate of SI-FCFS falls when N_2 is increased beyond 26 even though the probe rate increases (probe and transfer rate data are not shown).

RR Policy: For reasons given in Section 4.1, when RR node scheduling policy is used, the SI-RR policy performs significantly better than RI-RR policy. As shown in Figure 4.2, RI-RR performs slightly better than RI-FCFS policy. The SI-RR policy, however, exhibits fairly robust behaviour and provides the best performance among the four policies. The performance

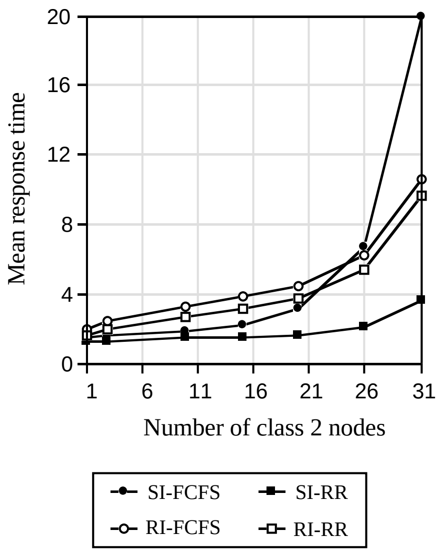


Figure 4.2 Performance sensitivity to degree of heterogeneity (N = 32 nodes, $N_1 = N - N_2$, N_2 is varied, $\mu_1 = \mu_2 = 1$, $C_{s1} = C_{s2} = 4$, $\lambda_1 = 0.1$, $\lambda_2 = 0.9$, $C_{a1} = C_{a2} = 4$, $T_m = 2$, $T_S = 2$, $T_R = 1$, $P_l = 3$, $P_I = 1$, transfer cost = 1%, $Q = 0.1$, CSO = 1% of Q)

superiority of RR policies can be attributed to high service time CV used in this set of experiments. Performance sensitivity to inter-arrival and service time CVs is discussed in the next two sections.

4.3. Sensitivity to variance in inter-arrival times

The impact of inter-arrival CV is shown in Figure 4.3. For both classes, the service time CV is 1. The inter-arrival time CV C_{ai} is set to the same value in both classes and is shown on the x-axis. The receiver-initiated policies are relatively more sensitive to the inter-arrival CV than the sender-initiated policies. The reason for this sensitivity is that increased CV implies clustered nature of job arrivals into the system; this results in reduced probe rate. For example, assume that $C_{si} = 0$ for both classes. If C_{ai} is also zero, each node invokes the probing process to locate a sender node after the completion of each job because local job arrival would be some time after the completion of the previous job (because of no variations in either service or inter-arrival times). Now increasing C_{ai} means clustered arrival of jobs. Suppose four 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. Although not shown here, probe rates decrease with increasing inter-arrival CV. On the other hand, clustered arrival of jobs fosters increased load sharing activity in sender-initiated policies. Thus the probe and job transfer rates increase with increasing inter-arrival CV (not shown). The sender-initiated policies tend to transfer more jobs than the receiver-initiated policies. For example, when C_{ai} is 4, the transfer rate of sender-initiated policies is 0.44 per node while that of receiver-initiated policies is 0.34.

For both sender- and receiver-initiated policies, the RR policy performs marginally worse than the FCFS policy because of low service time CV (which is 1 for these experiments). Note that the RR policy improves performance when there is high variance in service times.

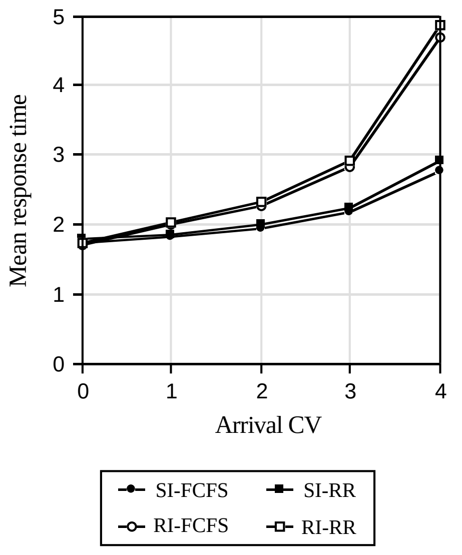


Figure 4.3 Performance sensitivity to inter-arrival time CV (N = 32 nodes, N_1 = 6, N_2 = 26, λ_1 = 0.1, λ_2 = 0.9, C_{a1} = C_{a2} = varied from 0 to 4, $\mu_1 = \mu_2$ = 1, C_{s1} = C_{s2} = 1, $T_m = T_S$ = 2, T_R =1, P_l = 3, P_I = 1, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

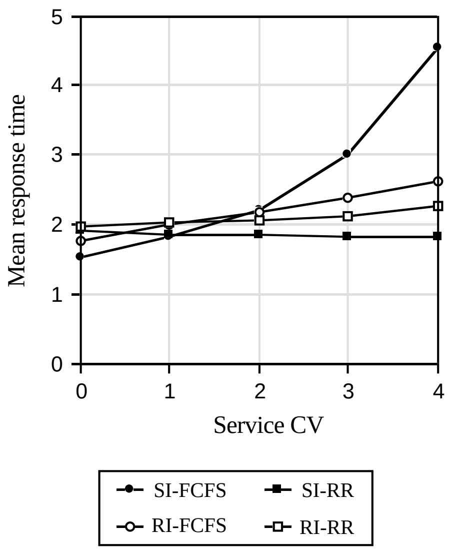


Figure 4.4 Performance sensitivity to service time CV (N = 32 nodes, N_1 = 6, N_2 = 26, λ_1 = 0.1, λ_2 = 0.9, C_{a1} = C_{a2} = 1, $\mu_1 = \mu_2$ = 1, C_{s1} = C_{s2} = varied from 0 to 4, $T_m = T_S$ = 2, T_R = 1, P_l = 3, P_I = 1, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

4.4. Impact of variance in service times

Figure 4.4 shows the impact of variance in service time. In this context, we should note that service time distribution of the data collected by Leland and Ott [12] from over 9.5 million processes has been shown to have a coefficient of variation of 5.3.

For this experiment, the inter-arrival time CV for both classes is fixed at 1. There are two main observations from this figure. When the non-preemptive FCFS node scheduling policy is used, sender-initiated policies exhibit more sensitivity to C_{si} than the receiver-initiated policies. At high C_{si} (for example, when C_{si} = 4) receiver-initiated policy (RI-FCFS) performs substantially better than the sender-initiated policy (SI-FCFS). The sensitivity of the FCFS policy to service time CV is an expected result because larger jobs can monopolize the processing power that results in increased probe rates (not shown). However, this increase in probing activity is useless because, at high system loads, the probability of finding a receiver node is small. At high C_{si}, RI-FCFS performs substantially better than SI-FCFS 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).

When the preemptive RR policy is used, sender-initiated policy provides the best performance unless the service time CV is low. Furthermore, both policies (SI-RR and RI-RR) exhibit negligible sensitivity to service time CV. This is mainly due to the preemptive nature of the RR scheduling policy. Even though not shown here, the probe rate associated with SI-RR policy decreases marginally with C_{si}. This is because higher C_{si} implies the presence of a large number of small jobs and a small number of large jobs. Because of its preference to complete smaller jobs, RR policy tends to move more jobs to the corresponding node's local job queue and thus initiates fewer load distribution activities. This tendency increases with increasing C_{si}. This, however, does not affect the load sharing capability as the job transfer rate is similar to that in the FCFS policy for high C_{si}. The probe rate of RI-RR increases marginally with C_{si}. This is due to the fact that an increase in C_{si} results in a large number of small jobs and, therefore, the probability of a node becoming idle increases.

4.5. Performance sensitivity to probe limit

Figure 4.5 shows the sensitivity of performance to probe limit. It can be seen that only the SI-RR policy is least sensitive to probe limit P_l.

For the receiver-initiated policies, changing the probe limit does not introduce significant performance differences between RR and FCFS scheduling policies. The data presented in Figure 4.5 show that the response time is extremely sensitive to the probe limit. For reasons explained in the previous sections, the RI-RR policy generates fewer probes and job transfer rates. As reported in [1], at higher system loads and for higher probe limits, receiver-initiated policies perform better than SI-FCFS policy. However, their performance is still worse than that of the SI-RR policy.

For the sender-initiated policies, SI-RR policy exhibits negligible sensitivity to the probe limit for P_l>2. The SI-FCFS policy results in substantial performance degradation if too small a probe limit is used. For reasons explained in Sections 4.1 and 4.4, the SI-RR policy results in fewer probes but transfers more jobs. As a result the SI-RR policy exhibits less sensitivity to probe limit.

4.6. Performance sensitivity to threshold

Figure 4.6 shows the the impact of threshold on the performance.

Sender-initiated policies: In general, increasing the threshold value decreases the scope of load sharing. This manifests in

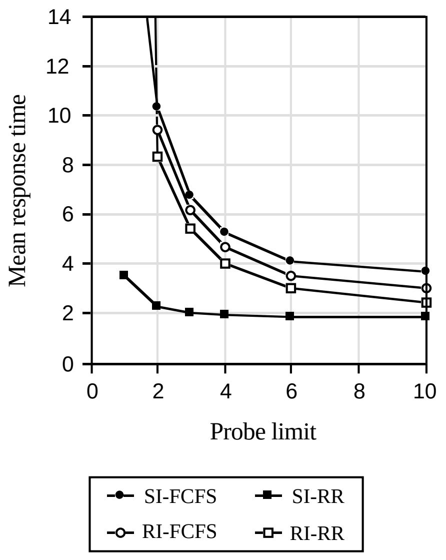


Figure 4.5 Performance sensitivity to probe limit (N = 32 nodes, N_1 = 6, N_2 = 26, λ_1 = 0.1, λ_2 = 0.9, $C_{a1} = C_{a2}$ = 4, $\mu_1 = \mu_2$ = 1, $C_{s1} = C_{s2}$ = 4, T_m = 2, T_S = 2, T_R = 1, P_l is varied from 1 to 10, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

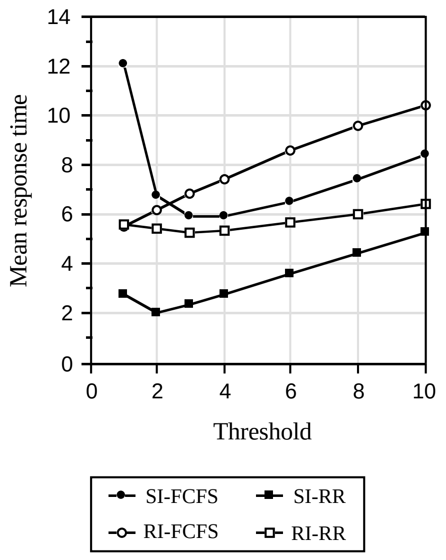


Figure 4.6 Performance sensitivity to the threshold parameter (N = 32 nodes, N_1 = 6, N_2 = 26, λ_1 = 0.1, λ_2 = 0.9, $C_{a1} = C_{a2}$ = 4, $\mu_1 = \mu_2$ = 1, $C_{s1} = C_{s2}$ = 4, T_m and T_S are varied, T_R = 1, P_l =3, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

decreasing probe rates as the threshold is increased. It should be noted that $T_m = T_s = \infty$ corresponds to a system with no load sharing. At $T_m = T_s$ =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 at high system loads. This, therefore, results in poor performance at high system loads as shown in Figure 4.6.

The SI-FCFS policy is more sensitive than SI-RR policy. Both sender-initiated policies show performance improvements initially and further increase in threshold would result in performance deterioration as shown in Figure 4.6. 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 (not shown). Further increase in the threshold value results in decreasing the number of attempts at load distribution. This causes reduced probe rates as well as job transfer rates.

Note that, when $T_m = T_s = 1$, the performance of the two sender-initiated policies is significantly different. The main reason for this behaviour is that, even though the threshold is 1, the job queue length could be higher than 1 due to (i) the transfer of local jobs to the local job queue because no receiver node had been found and (ii) the transfer of jobs that have been transferred from another node. Note that the policies dictate that a transferred job has to be processed at the destination node regardless of the state of the node at the time of its arrival.

Receiver-initiated policies: The performance of the RI-FCFS policy deteriorates with increasing threshold value T_m. The main reason for this is that the system moves towards diminished load sharing system and the service time CV is high. It is well-known that the FCFS scheduling discipline is sensitive to the service time CV. RI-RR policy exhibits least sensitivity to the threshold value. As in SI-RR policy, performance improves initially with the threshold value; further increases in threshold causes performance deterioration. However, SI-RR still provides better performance than RI-RR for the threshold values considered here. Since a threshold value of less than 3 yields the best performance, SI-RR maintains its performance superiority over the other three policies.

5. Performance of Type II Systems

In type II heterogeneous systems, nodes may have different processing rates in addition to seeing different job arrival rates as in the type I systems. We consider a system with N = 32 nodes consisting of two node classes with processing rates of μ_1=0.5 and μ_2=1. Due to space limitations, we will present only a sample of the simulation results.

5.1. Performance as a function of system load

This section presents the results as a function of offered system load. For this experiment, we have divided the 32 system nodes equally between the two node classes (i.e., $N_1 = N_2$ =16). The offered system load is maintained the same for both classes. That is, the arrival rate of class 1 is maintained at half of that for the class 2 nodes because class 2 nodes are twice as fast (i.e., $\lambda_1 = \lambda_2/2$). We have run several experiments to determine the best threshold value for each class. Even though the value varies slightly depending on the node scheduling algorithm, the threshold values of T_1 = 2 for the class 1 nodes and T_2 = 3 for class 2 nodes are found to be good choices. The results are presented in Figure 5.1. The inter-arrival and service time CVs for both classes are fixed at 4 (i.e., $C_{ai} = C_{si}$ = 4).

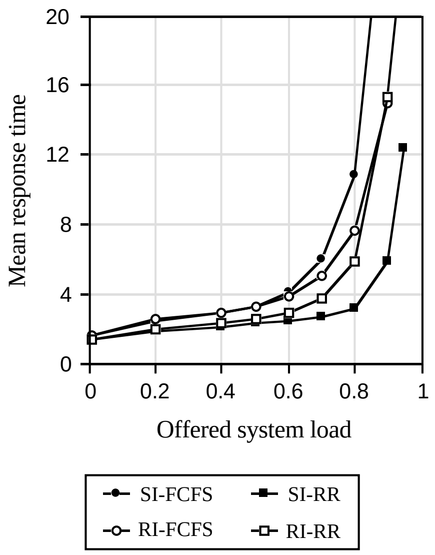


Figure 5.1 Performance sensitivity to offered system load (N = 32 nodes, $N_1 = N_2 = 16$, $\lambda_1 = \lambda_2/2$, λ_2 is varied, $C_{a1} = C_{a2} = 4$, $\mu_1 = 0.5$, $\mu_2 = 1$, $C_{s1} = C_{s2} = 4$, $T_{m1} = T_{S1} = 2$, $T_{m2} = T_{S2} = 3$, $T_{R1} = T_{R2} = 1$, P_l =3, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

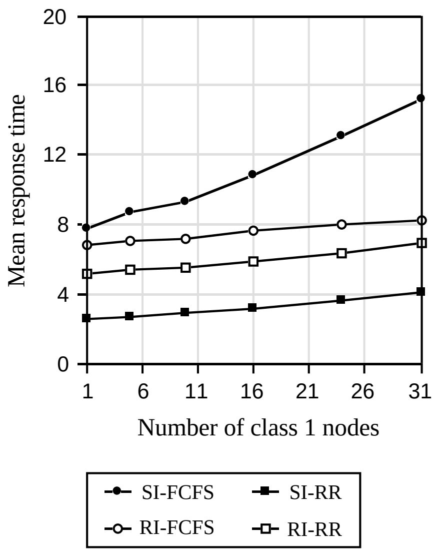


Figure 5.2 Performance sensitivity to the threshold parameter (N = 32 nodes, N_1 is varied, $N_2 = N - N_1$, $\lambda_1 = 0.4$, $\lambda_2 = 0.8$, $C_{a1} = C_{a2} = 4$, $\mu_1 = 0.5$, $\mu_2 = 1$, $C_{s1} = C_{s2} = 4$, $T_{m1} = T_{S1} = 2$, $T_{m2} = T_{S2} = 3$, $T_{R1} = T_{R2} = 1$, P_l =3, transfer cost = 1%, Q = 0.1, CSO = 1% of Q)

FCFS policy: When the node scheduling policy is FCFS, both sender-initiated and receiver-initiated policies perform the same until the offered system load is about 0.6. Beyond that the receiver-initiated policy performs significantly better than the sender-initiated policy. The performance difference between SI-FCFS and RI-FCFS also depends on the number of class 1 and class 2 nodes. If there are more class 1 nodes (which are slow nodes in our case), the performance difference increases. This issue is discussed in the next section.

Analysis of simulation data for each class suggests that, in SI-FCFS policy, more jobs are transferred from class 2 to slower class 1 nodes. Since the job arrival rate of class 2 nodes is twice that of the class 1 nodes and T_1 is 2 and T_2 is 3, the probability of finding a sender node in class 2 is higher than in class 1. Since the probing process is unbiased, more jobs are transferred to class 1 nodes. As a result, the response time of class 1 jobs is substantially higher than that of class 2 jobs.

In RI-FCFS policy, job movement across the class boundary is more balanced than in SI-FCFS policy. This is because class 2 nodes have a higher probability of being in the sender state than class 1 nodes. As a result of this balanced job movement, the difference between the response times of the two classes is smaller than in SI-FCFS policy.

RR policy: The SI-RR policy provides the best performance even in type II heterogeneous systems. As discussed in Section 4.1, when the preemptive RR policy is used, the probability of finding a node in the receiver state (i.e., below threshold) is high because RR gives priority to smaller jobs. Thus with the RR policy, probability of finding a node in the sender state decreases. Therefore, the SI-RR policy is more successful in finding a receiver node to transfer a job; on the other hand, RI-RR policy suffers because it is less successful in finding a sender node. The net effect of this is that job transfer rate of SI-RR is substantially higher than that of RI-RR policy. For example, when the offered system load is 0.8, the job transfer rate of SI-RR is 0.32 while that for the RI-RR is 0.2.

5.2. Impact of degree of heterogeneity

Figure 5.2 shows the response time as a function of number of class 1 nodes N_1. Note that the number of class 2 nodes (i.e., faster nodes) N_2 is given by $32-N_1$. Thus as we move from left to right in Figure 5.2, the number of slower nodes increases in the system. As a result all policies experience increase in response time. However, only the SI-FCFS policy exhibits more sensitivity to N_1 than the other three policies. This is because the sender-initiated policy finds it difficult to locate a receiver as there are fewer receivers than senders (because the system load is 80%). As a result, the SI-FCFS policy moves the system closer to no load sharing than the RI-FCFS policy. This coupled with the fact that there are more slower nodes as we move right in Figure 5.2 causes the response time increases much more quickly with SI-FCFS than RI-FCFS policy. The RR policies are less sensitive to the degree of heterogeneity because of their preemptive nature. It should be noted that in all cases the SI-RR policy gives the best performance.

6. CONCLUSIONS

We have discussed the impact of node scheduling policies on the performance of sender-initiated and receiver-initiated dynamic load sharing policies. We have considered two node scheduling policies - first-come/first-serve (FCFS) and round robin (RR) policies. We have also considered two types of heterogeneous systems and studied the impact of variance in inter-arrival times and job service times on the performance of these two node scheduling policies.

For the parameter values considered here, we have shown that:

When the non-preemptive FCFS node scheduling policy is used:

- receiver-initiated policies are highly sensitive to variance in the inter-arrival times (i.e., to clustered arrival of jobs);
- sender-initiated policies are relatively more sensitive to variance in job service times.
- sender-initiated policies are relatively more sensitive to the degree of heterogeneity than the receiver-initiated policy.

When the preemptive RR node scheduling policy is used:

- sender-initiated policy is less sensitive to system load, variance in the job service times, degree of heterogeneity for a given variance in the inter-arrival times;
- when the variance in service time is high, the sender-initiated policy provides the best performance among the policies considered in this study.

These observations are similar in nature to the conclusions reported for homogeneous distributed systems [1].

ACKNOWLEDGEMENTS

I thank Siu-Lun Au for his help with parts of the simulation program. I 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 Int. Conf. Dist. Computing Systems,* Vancouver, May 1995, pp. 484-492.

[2] S.P. Dandamudi and H. Hadavi, "Performance Impact of I/O on Sender Initiated and Receiver Initiated Adaptive Load Sharing in Distributed Systems", *Int. Conf. Parallel and Distributed Computing Systems*, Dijon, France, 1996, pp. 507-514.

[3] S.P. Dandamudi and M. Lo, “A Hierarchical Load Sharing Policy for Distributed Systems", *IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, Haifa, Israel, 1997.

[4] 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.

[5] 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.

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

[7] 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.

[8] 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.

[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] P. Krueger and R. Chawla, “The Stealth Distributed Scheduler,” *IEEE Int. Conf. Dist. Computing Systems,* 1991, pp. 336-343.

[11] 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.

[12] W. E. Leland and T. J. Ott, “Load Balancing Heuristics and Process Behavior,” *Proc. PERFORMANCE 86* and *ACM SIGMETRICS 86*, 1986, pp. 54-69.

[13] 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.

[14] M. Lo and S.P. Dandamudi, “Performance of Hierarchical Load Sharing in Heterogeneous Distributed Systems,” *Int. Conf. on Parallel and Distributed Computing Systems*, Dijon, France, 1996, pp. 370-377.

[15] 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.

[16] M. Mutka and M. Livny, “Profiling Workstation’s Available Capacity for Remote Execution,” *Proc Performance 87,* Brussels, Belgium, 1987, pp. 529-544.

[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,* Vol. C-34, 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.