Sensitivity Evaluation of Dynamic Load Sharing in Distributed Systems[1]

Sivarama P. Dandamudi
Centre for Parallel and Distributed Computing
School of Computer Science, Carleton University
Ottawa, Ontario K1S 5B6, Canada
sivarama@scs.carleton.ca

TECHNICAL REPORT TR-97-12

ABSTRACT

Load sharing improves performance of distributed systems by distributing load from heavily-loaded nodes to lightly-loaded nodes in the system. We consider two basic 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. In most previous studies, the first-come/first-served node scheduling policy has been used. In addition, analysis and simulations in these studies have been done under the assumption that service times and inter-arrival times of jobs are exponentially distributed. The behaviour of these policies is not clear when these assumptions are relaxed. We report the sensitivity of the performance of the sender-initiated and receiver-initiated policies to node scheduling policy, variance in service times, and variance in inter-arrival times. Our objective is to provide an intuitive understanding of the behaviour of these policies that transcends specific system and workload models and parameter values used.

Index Terms: Dynamic load sharing, Distributed systems, Heterogeneous distributed systems, Performance evaluation, Process scheduling.

1. INTRODUCTION

Performance of distributed systems depends on several aspects of system and workload characteristics. In distributed systems, load is not evenly distributed among the nodes due to statistical fluctuations. This is the case even when all nodes are of the same type, and each is subjected to the same average workload. Load sharing improves performance of distributed systems by distributing the system workload from heavily-loaded nodes to lightly-loaded nodes in the system.

Load sharing policies can be classified into three types: *static*, *dynamic* or *adaptive* policies. Static policies only use information about the average system behaviour. For example, jobs can be assigned to nodes in a cyclic fashion [Shi92]. The chief advantage of the static policies is their simplicity. There is no need to collect system state information. The disadvantage of these policies is that they cannot respond to changes in system state; therefore the performance improvement is limited.

Dynamic policies, on the other hand, use current or recent system state information in making load distribution decisions. These policies react to system state changes dynamically. Therefore,

---

[1] An edited vesion of this paper will appear in *IEEE Concurrency*.
This report can be obtained from `http://www.scs.carleton.ca`

substantial performance improvement is possible over and above the improvement provided by the static policies. However, this additional performance benefit is obtained at a cost - the cost of collecting and maintaining the system state information. As long as we keep these overheads within reasonable limits, dynamic policies provide improved performance over static policies. Two classes of policies that belong to this group are the *sender-initiated* and *receiver-initiated* policies. 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. More details on these two policies are given later.

The final category of adaptive policies changes the load sharing policy and/or parameters of the policy depending on system and workload conditions. For example, an adaptive policy might use a sender-initiated policy at low to moderate system loads and a receiver-initiated policy at high system loads [Shi92].

Several load sharing policies have been proposed in the literature. Our focus is on the sender-initiated and receiver-initiated dynamic load sharing policies. There are two main reasons for considering these two classes of policies. First, several implementations use load sharing policies belonging to these two classes. The other reason is that a majority of the adaptive policies, as mentioned in the last paragraph, use these two classes of policies. Thus, it is important to understand how these two basic classes of policies behave under various system and workload conditions. We use “node” and “workstation” interchangeably.

Our objective is to provide intuition on the behaviour of the sender-initiated and receiver-initiated policies for various system and workload characteristics. Specifically, we report performance sensitivity of these policies to three factors: node scheduling policy, variance in inter-arrival times of jobs, and variance in job service times. We present results for homogeneous as well as heterogeneous systems. In homogeneous systems, all nodes are of the same type and the workload serviced by each node in the system exhibits the same average behaviour. In heterogeneous systems, nodes as well as the workload serviced by these nodes could be different.

Our emphasis is on providing an intuitive explanation of the underlying behaviour of these two policies rather than the absolute performance values. We have used several simulation experiments to corroborate the conclusions reported even though only a sample of these simulation results are included. We contend that such an understanding of these basic policies would benefit many people working in this area in devising new load sharing policies. Furthermore, with the growing interest in using networks of workstations as parallel virtual machines, the intuition provided would also benefit in adapting the load sharing policies proposed for distributed systems to parallel virtual machines.

2. DYNAMIC LOAD SHARING POLICIES

Two important components of a dynamic policy are a transfer policy and a location policy [Eag86a]. 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 queue length, CPU utilization, the amount of available memory, etc. have been proposed. 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 [Kun91].

We will now describe the two dynamic load sharing policies that have been used in our simulation experiments. Our load sharing policies are slightly different from those reported in the

literature [Eag86a/b, Dik89]. These subtle changes are motivated by the following aspects of locally distributed systems. First, workstations use a round robin (RR) scheduling policy for efficiency and performance reasons. In the RR 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. Round robin scheduling, therefore, eliminates a particularly large service demand job from monopolizing the processor. However, each preemption would cause a context switch overhead.

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 [Eag88]. Load sharing facilities like Utopia [Zho93] will not consider migrating active processes. However, process migration is desirable for three practical reasons.

(i) When the owner of a workstation claims control of his/her workstation, migrating the job to another workstation is desirable to restarting it on another workstation. Most example load distribution packages described in Appendix B support process migration. The Cray NQE does not support process migration but uses a job restart strategy. The major disadvantage of process migration is that large files representing the state information of jobs will have to be moved on the network, which may have an impact on users of the network [Bak96].

(ii) When a workstation fails, the job on that workstation can be restarted from the last checkpointed position. Checkpointing saves the state of a job at regular intervals. A drawback of checkpointing is that it requires hardware to implement. Even for small jobs, checkpointing at regular intervals requires large amount of disk space at each workstation [Bak96]. Except for the NQE system, all systems mentioned in Appendix B support checkpointing.

(iii) Process migration is also useful to achieve better load balancing among the workstations. For example, if the status of a workstation changes from lightly-loaded to heavily-loaded due to an increase in locally generated background load, it is useful to move the job to another workstation.

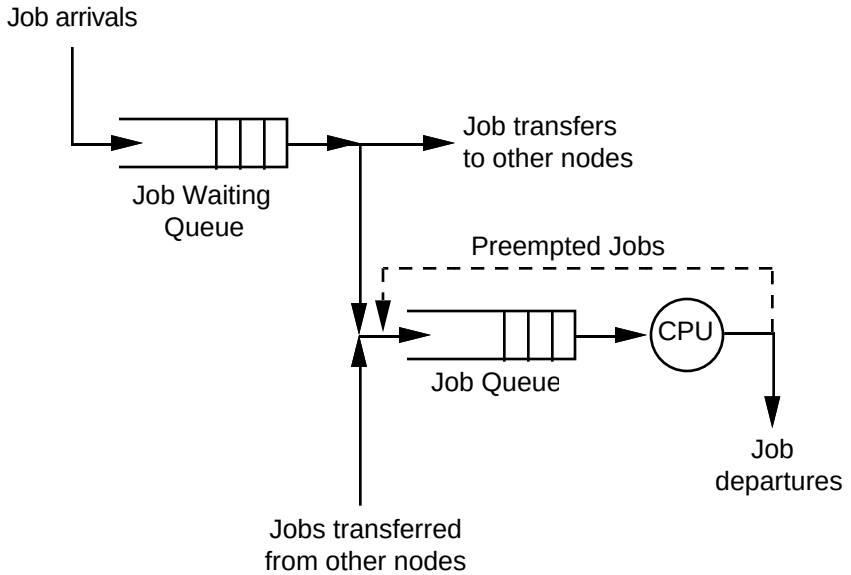

Figure 2.1 Logical structure of a node (Job preemptions, shown dashed, are used only by the Round Robin node scheduling policy)

In this paper, we do not consider node failures, node ownerships, or background load - the three conditions for which process migration is useful. As a result process migration is not considered in our model. We assume that each node has two queues to hold jobs: a *job queue*, which is a queue of all jobs (i.e., active jobs) to be executed locally and a *job waiting queue* that holds those jobs that are waiting to be assigned to a node (see Figure 2.1). Note that only the jobs in the job waiting queue are eligible for load distribution.

Implementations of load sharing policies use either a local job waiting queue at each node or a single central job waiting queue for the entire cluster of nodes. As discussed in Appendix B, Condor and Batrun use local job waiting queues while Codine, LSF, and NQE maintain a central job waiting queue. The implementation described in [Dik89] has a long-term scheduler that maintains a queue of jobs before they are submitted to the UNIX operating system for execution. They have implemented the long-term scheduler for the same reasons that we have described here (i.e., to avoid process migration).

In both sender-initiated and receiver-initiated policies, we use a threshold-based transfer policy. When a new job arrives at a node, the transfer policy looks at the job queue length of the node. It transfers the new job to the job queue (i.e., for local execution) if the job queue length is less than a predetermined threshold *T*. Otherwise, the job is placed in the job waiting queue of the node for possible remote execution. The location policy, when invoked, performs the node assignment.

When the round robin node scheduling policy is used, a multiprogramming level T_M is used. Multiprogramming level is the number jobs among which the processor is time shared. Since we assume that a processor is shared among the jobs in the job queue, the threshold *T* is equal to the multiprogramming level T_M. The desired value of T_M depends on several factors including resources available at a node, desired performance level, resource requirements of jobs, and mode of operation. For example, if a typical job requires half of the available memory at a node for the expected level of performance, a multiprogramming level of two can be used. Mode of operation can be dedicated or shared. In the dedicated mode, only a single job is run, which implies a multiprogramming level of one. For example, IBM's LoadLeveler provides three dedicated classes: half_hour_dedicated, two_hour_dedicated, and offpeak_dedicated. The first two classes provide the user with dedicated use of the node for 0.5 and 2 hours, respectively. The offpeak_dedicated class allows 12-hour jobs to be scheduled on weeknights and 72-hour jobs on the weekend.

Next we discuss the two location policies considered in this paper.

Sender-initiated policy

When a new job arrives at a node, the transfer policy described before decides whether the job is placed in the job queue or the job waiting queue. If the job is placed in the job waiting queue, the job is eligible for transfer to another node 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 the sender threshold T_S, which is the equal to the multiprogramming level T_M. If such a node is found, the job is transferred to that node for remote execution. The transferred job is directly placed in the destination node's job queue (see Figure 2.1). 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 has changed since the probing.

Receiver-initiated policy

When a new job arrives at node *j*, the transfer policy places the job either in the job queue or in the job waiting queue of node *j* as described before. The location policy is invoked by the nodes at job completions. The location policy of node *j* 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 *j* is less than the receiver threshold T_R, it initiates the probing process to locate a node *i* 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 *i* will be transferred to the job queue of node *j*. In this paper, as in the previous literature [Eag86b, Shi92], 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 spending time on 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 [Mut91]. Thus the probability of a workstation being idle is high.

What is known?

A common thread among the previous studies, with a few exceptions (e.g., [Zou88] and [Dik89]), is they assume the non-preemptive FCFS node scheduling policy. Furthermore, analyses and simulations in these studies have been done under exponentially distributed job service times and inter-arrival times of jobs.

We use mean response time to represent performance of the system. Response time of a job is defined as the time between the job completion time and its arrival time. We used one time unit as the average job service time, and this is the unit used for the mean response time. We use the terms "performance" and "mean response time" interchangeably.

Figure 2.2 shows the performance of the sender-initiated and receiver-initiated policies in a 32-node system as a function of system utilization. The system utilization is defined as the average system utilization due to the execution of jobs only. In computing the system utilization, we do not include the overhead activities such as job transfers, sending and receiving probe messages. Thus, the actual system utilization is higher than the system utilization used on the x-axis. For comparison purposes, we have also included the performance under no load sharing. These results are for the FCFS node scheduling policy. Both service times and inter-arrival times are exponentially distributed. As mentioned, the average job service time is one time unit. All other parameters are set to their default values given in Appendix A. Since the service times and inter-arrival times are exponentially distributed, performance under no load sharing can be obtained analytically by using the M/M/1 queueing model. The mean response time of such a system is given by $\frac{1}{\mu-\lambda}$ where μ is the mean service rate of a node and λ is the mean arrival rate of jobs at a node. The results for the sender-initiated and receiver-initiated policies are obtained by simulation.

We can see that both dynamic load sharing policies provide substantial performance improvements over no load sharing. This figure also demonstrates the relative performance of the two dynamic policies. The sender-initiated policy performs better than the receiver-initiated policy at low to moderate system loads. Why? 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, the receiver-initiated policy is better because it is much easier to find a heavily-loaded node at these system loads. This relative performance has prompted several researchers to devise adaptive policies that behave like the sender-initiated policy at low to moderate system loads and like the receiver-initiated policy at high system loads [Shi92].

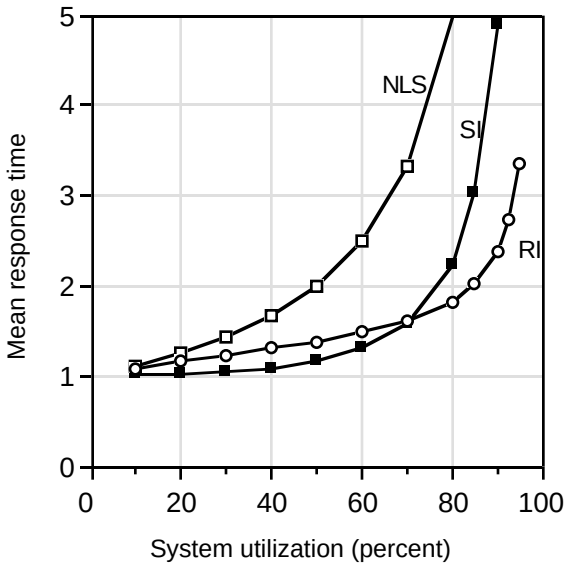

Figure 2.2 Performance of sender-initiated (SI), receiver-initiated (RI), and no load sharing (NLS) policies as a function of system utilization.

Most previous studies have used exponential inter-arrival and service times for jobs as we have done for the data presented in Figure 2.2. However, empirical data collected by Zhou [Zho88] and others suggest that variance in both inter-arrival times and service times is higher than that of the exponential distribution. We express variance as the ratio of the standard deviation to mean, which is called the coefficient of variation (CV). Thus, the higher the CV, the larger the variance. The exponential distribution has a coefficient of variation of 1, implying that the standard deviation is equal to the mean. The Berkeley data collected by Zhou indicates that the coefficient of variation of inter-arrival times is 2.65 and that of service times is 12.83!

The impact of service time variance is studied by Zhou [Zho88] and Dikshit et al. [Dik89]. However, there has been no systematic study to look at the impact of the variances in inter-arrival times and service times of jobs as well as the node scheduling policy used. This lack of information has motivated the research reported in this paper.

3. SENSITIVITY RESULTS FOR THE HOMOGENEOUS SYSTEM

We now discuss the performance sensitivity of the sender-initiated and receiver-initiated policies to the three factors mentioned. In this section we consider only homogeneous distributed systems. In these systems, all nodes are of the same type and the workload serviced by each node in the system exhibits the same average behaviour. Heterogeneous systems are discussed in the next section.

In total we present five observations. For each observation, we give a short statement summarizing the observation and then follow it up with a discussion that gives a sample quantitative evidence in support of the observation. Note that, while the sample data were obtained for a particular set of parameter values, the general trends reported are valid for other system and workload parameters. The focus of the discussion is on the intuitive reasoning for the behaviour. While changes in system and workload models might change the absolute value of the mean response times reported here, the overall behaviour and the intuition developed will remain valid for a wide range of system and workload parameters. For example, as stated in the previous section, it is widely known that the sender-initiated policy performs better at low to moderate

system loads and the receiver-initiated policy performs better at high system loads. This general intuition is true (under the conditions stated in the last section), and the actual cross-over point (which is 0.7 in Figure 2.2) might vary depending on the system and workload assumptions made. We are after that kind of intuition and not the absolute performance numbers.

Observation 1: The receiver-initiated policy is more sensitive to the variance in job inter-arrival times than the sender-initiated policy.

Sample supporting data: In support of this claim, we present the data from the simulation experiments conducted on a 32-node system in Figure 3.1. This figure shows the mean response time when the system utilization is fixed at 80% (i.e., mean inter-arrival rate λ is 0.8 and mean service time $1/\mu$ is 1) and service time CV at 1. All other parameter values are set to their default values given in Appendix A. For our discussion here, we need to consider only the RI and SI lines (and ignore the third line, which will be discussed in Observation 2). From this plot it is obvious that the receiver-initiated policy is very sensitive to the variance in inter-arrival times. In comparison, the sender-initiated policy exhibits a more robust behaviour. For example, when the inter-arrival CV is varied from 0 to 4, the mean response time of the receiver-initiated policy increases by a factor of 7.5 whereas the corresponding value for the sender-initiated policy is 1.75.

Explanation: To understand the sensitivity of the receiver-initiated policy, we should note that higher inter-arrival CV implies that job arrivals at each node in the system are clustered in nature; the higher the CV the more clustered the job arrival process is. Clustered arrivals imply that there are large gaps between job arrivals as the average job arrival rate remains the same independent of the variance in the inter-arrival times. This has two serious performance implications for the receiver-initiated policy.

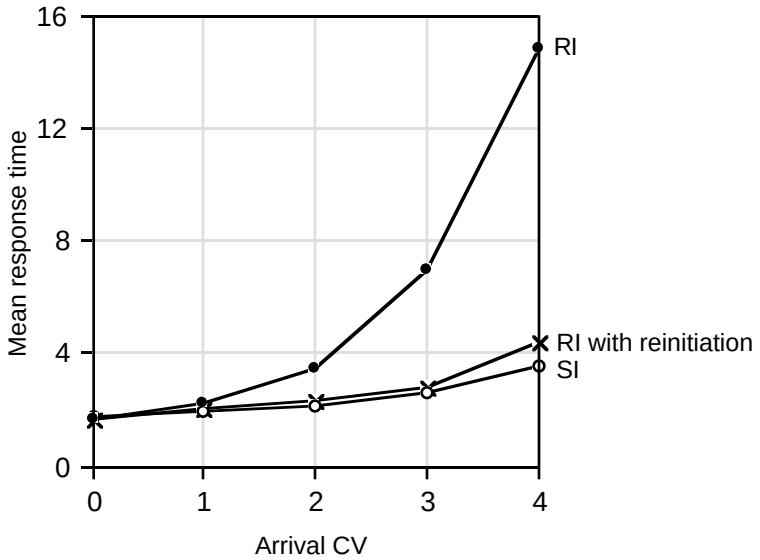

Figure 3.1 Sensitivity of performance of the sender-initiated (SI), receiver-initiated (RI), and modified receiver-initiated (RI with reinitiation) policies to variance in inter-arrival times of jobs. The node scheduling policy is the non-preemptive FCFS policy. The results for the preemptive round robin policy are similar and are not shown.

(i) Recall that, under the receiver-initiated policy, a node makes an attempt at load distribution every time the node is in receiver state when it completes a job. If that attempt fails to locate a sender node, it remains idle until a local job arrives at the node. When we increase the variance in job inter-arrival times, we introduce large gaps between job arrivals. Such large gaps lead to more idle time for the nodes in the system and moves the system towards no load sharing, which results in performance deterioration. It has been recognized that, for similar reasons, the receiver-initiated policy suffers in performance if the load is not homogeneous (e.g., if only a few nodes are generating the system load) [Shi92]. This adverse impact on the performance of the receiver-initiated policy can be reduced by reinitiating the load distribution activity after the elapse of a predetermined time if the node is still idle. The impact of this strategy is further discussed in Observation 2.

(ii) The second problem is that the clustered arrival of jobs into the system reduces the number of attempts that a receiver-initiated policy makes at load distribution. To illustrate this effect let us assume that there is no variance in job service times. That is, every job that is coming into the system has exactly the same service time requirement. Let us further assume that there is no variance in job inter-arrival times. In this case, each node under the receiver-initiated policy 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 either in service times or in inter-arrival times). Now increasing inter-arrival CV means clustered arrival of jobs. Suppose that four jobs arrive into the system as a cluster. Then, the node would not initiate any load sharing activity at least until it completes these four jobs. That is, the node would initiate load sharing only if no job arrives at this node within the next $4/\mu$ time units after the arrival of the first job in the four job cluster. Thus, increasing the inter-arrival CV pushes the system based on the receiver-initiated policy towards a no load sharing system.

In contrast, both these characteristics - clustered arrival of jobs and long gaps between the arrivals - of variance in inter-arrival times actually help improve load sharing in the sender-initiated policy. The clustered arrival of jobs fosters increased load sharing activity as the probability of finding a node in sender state increases with the clustered arrival of jobs into the system. For example, if four jobs arrive at a node as a cluster, the node attempts to distribute the jobs above the threshold limit of two in our example. Thus, the more clustered the arrival process is, the more attempts the sender-initiated policy makes at load distribution. We have, however, stated before that the sender-initiated policy does not perform well under high system load (which is the case in Figure 3.1 where the system utilization is 80%) because finding a receiver is difficult at these system loads. This is where the second characteristic - long gaps between arrivals - helps the sender-initiated policy in successfully distributing the load. We can see that, when there are long gaps between arrivals, the probability of a node moving into the receiver state increases. Thus, the probability of the sender-initiated policy locating a receiver increases with increasing variance in inter-arrival times.

Observation 2: The sensitivity of the receiver-initiated policy to variance in inter-arrival times can be reduced substantially by attempting periodic load distribution.

Sample supporting data: The data are presented in Figure 3.1 to support this claim. In the modified receiver-initiated policy, as in the original receiver-initiated policy, the receiver node attempts to find a sender node. If that attempt fails, the original policy waits for a local job to arrive whereas the modified policy attempts to locate a sender after the elapse of the reinitiation period δ if the node remains a receiver during the reinitiation period. The data presented in Figure 3.1 used a

reinitiation period equal to the average job service demand (i.e., $\delta = 1$). We can see substantial performance improvements with periodic reinitiation of load distribution (see "RI with reinitiation" line in Figure 3.1) compared to the original receiver-initiated policy (the "RI" line). For example, when the inter-arrival CV is 4, the response time reduces from about 15.5 to 4.6 when reinitiation is used. However, these values are still higher than those obtained for the sender-initiated policies as shown in Figure 3.1.

Explanation: We have discussed two main reasons for the sensitivity of the receiver-initiated policy to the variance in inter-arrival times (see Observation 1 explanation). What we have done with reinitiation is to reduce the impact on performance due to the first reason. Clearly, the smaller the reinitiation period, the better the performance. More details on the impact of the reinitiation period can be found in [Dan95]. However, reduced reinitiation period increases load distribution overhead as it causes more probing activity.

Still, the sender-initiated policy performs better then the receiver-initiated policy - particularly at high inter-arrival CV. The advantage of the sender-initiated policy is that the instant a node becomes overloaded due to clustered arrival of jobs, it initiates load distribution. The receiver-initiated policy, on the other hand, will only attempt load distribution either at the job completion time or periodically every reinitiation period δ if it remains a receiver.

Since reinitiation improves the performance of the receiver-initiated policy substantially, in the remainder of this paper we consider only this receiver-initiated policy with a reinitiation period of 1.

Observation 3: When the non-preemptive FCFS node scheduling policy is used, sender-initiated policy exhibits more sensitivity to variance in service times than the receiver-initiated policy.

Sample supporting data: Figure 3.2 shows the mean response time of the sender-initiated (SI-FCFS) and receiver-initiated (RI-FCFS) policy when the node scheduling policy is FCFS. Note that the multiprogramming level is not required when implementing the FCFS node scheduling policy. The data were obtained by simulation for a 32-node system. The system utilization is maintained at 80% (i.e., $\lambda = 0.8$). The mean job service time is 1 (i.e., $1/\mu = 1$) and inter-arrival times are exponentially distributed (i.e., inter-arrival time CV is 1). All other parameters are set to their default values given in Appendix A. The RI policy uses a reinitiation period δ of 1. The notation used is that the first two letters identify the load sharing policy and the letters after the hyphen identify the node scheduling policy. Thus, SI-FCFS represents the performance of the sender-initiated policy when the non-preemptive FCFS node scheduling policy is used.

This figure clearly demonstrates the fact that the sender-initiated policy is more sensitive to the variance in service times than the receiver-initiated policy. For example, when the service time CV is varied from 1 to 4, the mean response time of the sender-initiated policy changes from about 2 to 6 while the mean response time of the receiver-initiated policy changes from about 2 to 2.7.

Explanation: Note that increasing variance in service times implies that there is a large number of small jobs and a small number of large jobs. We use large and small jobs to refer to the service demand requirements of the jobs. The sensitivity of the non-preemptive FCFS scheduling policy to service time CV is an expected result because larger jobs can monopolize the processing power. For example, assume that a node gets three jobs - the first job is a large one and the remaining two are small ones. Assuming that the node is idle before the arrival of these three jobs, the first job starts receiving service and the second job will have to wait until the node completes the first large job. When the third job arrives at the node, it places the node above the threshold T_S of 2 that we are using here. Therefore, the node initiates probing to locate a receiver node. Thus, in the sender-

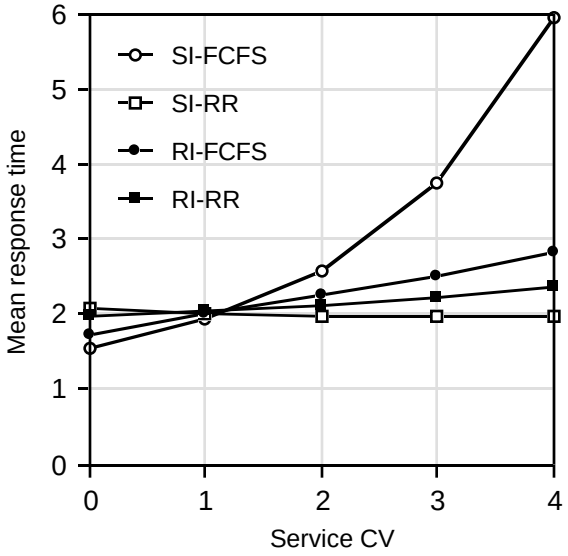

Figure 3.2 Sensitivity of performance of the sender-initiated (SI) and receiver-initiated (RI) policies to variance in service times of jobs. Note that RI represents performance of the receiver-initiated policy with reinitiation.

---

initiated policy, increased variance in service times results in increased probing activity. However, this increase in probing activity is useless because, at high system loads, the probability of finding a receiver node is small. When the service time CV is high, the receiver-initiated policy performs substantially better than the sender-initiated policy 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 better performance and in reduced sensitivity to service time variance.

Observation 4: When the preemptive round robin (RR) node scheduling policy is used, the receiver-initiated policy exhibits more sensitivity to variance in service times than the sender-initiated policy.

Sample supporting data: Figure 3.2 shows the mean response times of the sender-initiated (SI-RR) and receiver-initiated (RI-RR) policies when the round robin node scheduling policy is used. It can be seen from the data presented in this figure that the receiver-initiated policy is more sensitive to the variance in service times. On the other hand, the sender-initiated policy exhibits negligible sensitivity to the variance in service times. For example, when the service time CV is varied from 1 to 4, the mean response time of the sender-initiated policy remains constant at approximately 2 whereas the mean response time of the receiver-initiated policy changes from about 2 to 2.5.

Explanation: We can observe from Figure 3.2 that both load sharing policies perform better with RR policy than with the FCFS policy. This is mainly due to the preemptive nature of the RR scheduling policy. Between these two load sharing policies, the receiver-initiated policy is more sensitive to the service time variance. The sensitivity of the receiver-initiated policy increases if reinitiation is not used.

Recall that higher variance in service times implies the presence of a large number of small jobs and a small number of large jobs. Because of the preference given by the RR policy to smaller jobs, the RR policy tends to move more jobs to the corresponding node's local job queue. For

example, assume that a node has one large job at the front of the queue and one small job. Under the FCFS policy, the job queue length would be two until it completes the large job. When the RR policy is used, the node completes the smaller job first by sharing the processor and thus reduces the job queue length to one after that. Thus the probability of this example node being in the sender state decreases when the node scheduling policy is changed from the FCFS policy to the RR policy.

This aspect of the system behaviour benefits the sender-initiated policy in two ways. First, because the probability of a node being in the sender state decreases, there will be a reduced number of attempts at load distribution. Remember that, at high system load, the sender-initiated policy is not as successful in finding a partner to share the load as the receiver-initiated policy. Second, and more importantly, the probability of a node being in the receiver state (i.e., below the threshold value of 2 in our case) increases with the use of the RR policy. This has a positive benefit for the sender-initiated policy as the probability of finding a partner increases. For precisely the same reasons, the receiver-initiated policy finds it hard to locate a sender node. Thus, this policy exhibits relatively more sensitivity to the service time CV than the sender-initiated policy.

Observation 5: When the preemptive RR policy is used, the sender-initiated policy provides the best performance when there is high variance in service times and/or inter-arrival times.

Sample supporting data: Figure 3.3 presents the mean response times of the sender-initiated and receiver-initiated policies under the two the node scheduling policies. All results are obtained by simulation for a system with 32 nodes. The system utilization is varied by varying the average job arrival rate λ. The mean job service time is fixed at 1 (i.e., $1/\mu = 1$). The service time and inter-arrival time CVs are set at 4. The RI policy uses a reinitiation period δ of 1. All other parameters are set to the default values given in Appendix A. It can be seen from the data presented in Figure 3.3 that the sender-initiated policy performs best when the RR node scheduling policy is used (see the SI-RR line).

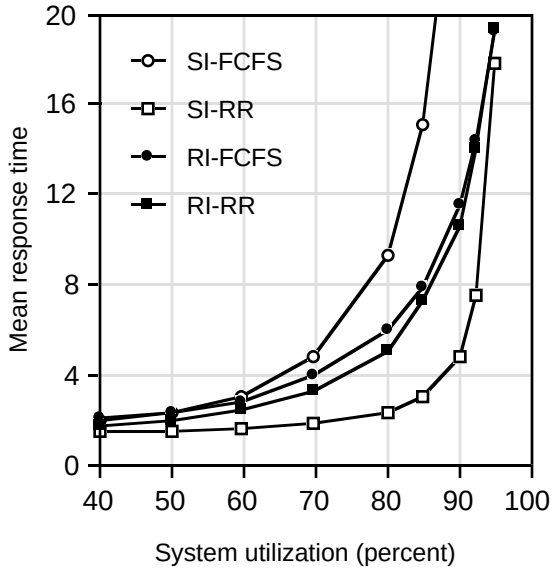

Figure 3.3 Sensitivity of performance of the sender-initiated (SI) and receiver-initiated (RI) policies to system utilization. The receiver-initiated policy uses a reinitiation period δ of 1.

Explanation: The data presented in Figures 3.1-3.3 show the interaction among the three factors that we are considering here: the node scheduling policy (FCFS versus RR), variance in service times, and variance in inter-arrival times. We can see from these figures that the node scheduling policy has the greatest impact on the performance of the sender-initiated policy. This is mainly because of the reduced sensitivity to the service time variance when the scheduling policy is changed from FCFS to RR. See Observation 4 for an explanation. Due to this sensitivity, the sender-initiated policy gives the best and worst performance when the node scheduling policy is RR and FCFS, respectively.

The receiver-initiated policy is relatively less sensitive to the scheduling policy. Remember that the receiver-initiated policy uses reinitiation. Therefore, as we can see from Figure 3.1, this policy is not as sensitive as the original receiver-initiated policy. Furthermore, the receiver-initiated policy is not as sensitive to the scheduling policy as shown in Figure 3.2.

4. PERFORMANCE IN HETEROGENEOUS SYSTEMS

In the last section, we presented several observations in the context of homogeneous distributed systems. In these systems, nodes in the system are all of the same type; furthermore, each node services, on average, a similar type of workload. Distributed systems are often heterogeneous in nature. We can identify two types of heterogeneity in distributed systems - workload heterogeneity and system heterogeneity. In the first category, even if the nodes are all the same, the workload serviced by these nodes could be different. For example, mean job arrival rate at some nodes may be higher and/or mean job service demand might be different at different nodes. In the second category, the nodes themselves might not be the same. For example, processing rates of nodes can differ from node to node in the system. In fact, systems might have a combination of these two types of heterogeneity.

Heterogeneous systems introduce additional parameters that make performance evaluation much more difficult compared to homogeneous systems. To simplify, we consider two types of heterogeneous systems as in [Mir90]. 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 classes of job arrivals, each with a different average job arrival rate. In type II systems, the node processing rates are also different. We consider two classes of nodes in type II systems - a fast node class and a slow node class. Nodes in each node class in type II system can have different job arrival rates and service rates. Since the results for the heterogeneous system are qualitatively similar to those for the homogeneous system, our discussion is rather brief and focuses only on the differences between the homogeneous and heterogeneous systems.

Performance in Type I Systems

The observations that we have made about the performance sensitivity of the sender-initiated and receiver-initiated policies in homogeneous systems are valid even for heterogeneous systems. However, some policies might exhibit more sensitivity in heterogeneous systems to system and workload parameters than in homogeneous systems. For example, the receiver-initiated policy exhibits more sensitivity to inter-arrival time variance even with a reinitiation period δ of 1 [Dan97]. To give an idea as to the behaviour of these two dynamic load sharing policies in a heterogeneous system, we present Figure 4.1.

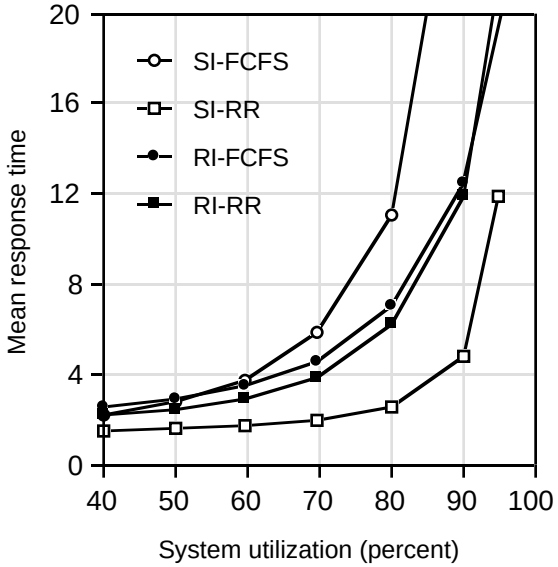

Figure 4.1 Performance of the sender-initiated and receiver-initiated policies in a type I heterogeneous system. The x-axis gives the system utilization on class 2 nodes. The receiver-initiated policy uses a reinitiation period δ of 1.

Figure 4.1 shows the mean response time as a function of system utilization of class 2 nodes. The 32 system nodes are divided into two classes: 6 class 1 nodes and 26 class 2 nodes. The class 1 nodes are assumed to be operating at a high system utilization of 90% (i.e., the average job arrival rate of class 1 nodes λ_1 is fixed at 0.9). Note that the offered system load for class *i* is given by λ_i/μ_i where λ_i and $1/\mu_i$ are the average job arrival rate and service times of class *i* nodes, respectively. Since we assume that $\mu_1 = \mu_2 = 1$ for the data presented in Figure 4.1, system utilization of class *i* nodes is equal to λ_i. Service time and inter-arrival time CVs for both classes are set at 4. All other parameters are set to their default values given in Appendix A. Comparing Figures 3.3 and 4.1, we can see that the relative performance of the policies is qualitatively the same. Performance sensitivity to other parameters is given in [Dan97].

Performance in Type II Systems

In type II heterogeneous systems, nodes have different processing rates in addition to seeing different job arrival rates as in type I systems. We consider a system with 32 nodes consisting of two node classes with processing rates of class 1 nodes μ_1=0.5 and class 2 nodes μ_2=1. That is, a job that takes a unit of time on a class 2 node requires twice that on a class 1 node. For this experiment, we have divided the 32 system nodes equally between the two node classes. The system utilization 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 (i.e., $\lambda_1 = \frac{\lambda_2}{2}$) because class 2 nodes are twice as fast. Service time and inter-arrival time CVs for both classes are set at 4. All other parameters are set to their default values given in Appendix A.

We have run several experiments to determine the best threshold value for each class of nodes. Even though the value varies slightly depending on the node scheduling policy, threshold values of 2 for class 1 nodes and 3 for class 2 nodes are found to be good choices. The results for the type II

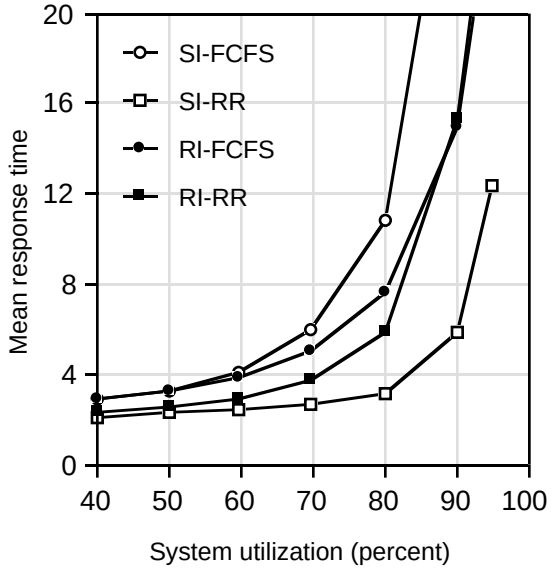

Figure 4.2 Performance of the sender-initiated and receiver-initiated policies in a type II heterogeneous system. The system utilization given on the x-axis is the same for both classes. The receiver-initiated policy uses a reinitiation period δ of 1.

heterogeneous system are presented in Figure 4.2. Again, the relative performance of the polices is qualitatively similar and confirms our earlier analysis presented in the context of homogeneous system model. It has been shown in [Dan97] that varying the distribution of nodes into these two classes does not change the relative performance of these policies. The absolute values, however, might change. For example, the performance difference between SI-FCFS and RI-FCFS policies increases if there are more class 1 nodes, which are slow nodes in our case.

5. SUMMARY

We discussed performance sensitivity of the sender-initiated and receiver-initiated dynamic load sharing policies to three factors: node scheduling policy (non-preemptive FCFS versus preemptive round robin), variance in service times, and variance in inter-arrival times. We provided five observations for homogeneous distributed systems in which the nodes are all similar and the workload they handle is also the same on the average. Our emphasis is on the underlying reasons for the sensitivity that these two policies exhibit to the three factors considered. Note that we do not consider node failures, node ownerships, and locally generated background load. Thus, process migration is strictly not required for performance improvement. For the system model and load sharing policies we considered, the following observations can be made:

(i) When non-preemptive FCFS node scheduling policy is used, the receiver-initiated policy is more sensitive to variance in inter-arrival times than the sender-initiated policy, and the sender-initiated policy is relatively more sensitive to variance in job service times. Results presented in [Dan96] indicate that this observation is valid even when I/O requirements of jobs are considered. The sensitivity of the receiver-initiated policy can be reduced by attempting periodic load distribution.

(ii) When the preemptive round robin node scheduling policy is used, the sender-initiated policy provides better performance than the receiver-initiated policy (even at high system loads) if

there is high variance in service times. This observation is in contrast to the prevailing intuition based on the FCFS node scheduling policy.

We have also noted that these observations are valid for heterogeneous systems.

The results presented in this paper are useful in at least two ways. First, a better understanding of the two basic dynamic load sharing policies is helpful in devising adaptive policies that take into account node scheduling policy and variances in service and inter-arrival times. Second, with the emergence of networks of workstations as a viable alternative to expensive parallel systems (e.g., multiprocessors and multicomputers), we can adapt some of the dynamic and adaptive load sharing policies, which have been extensively studied in the context of distributed systems, to these systems. The workload of these systems introduces new challenges to load sharing. For example, synchronization and inter-process communication are just two factors that can complicate the behaviour of load sharing policies for these systems. Further research is needed in this area. Also, the impact of process migration on the relative performance of the sender-initiated and receiver - initiated policies needs further study.

Acknowledgements

The author gratefully acknowledges the financial support provided by the Natural Sciences and Engineering Research Council and by Carleton University. I also acknowledge the contribution of Sui-Lun Au towards developing the initial simulation model for the load sharing policies discussed in this paper.

REFERENCES

[Dan95] S. P. Dandamudi, “Performance Impact of Scheduling Discipline on Adaptive Load Sharing in Homogeneous Distributed Systems,” *IEEE Int. Conf. Dist. Computing Systems,* 1995, pp. 484-492.

[Dan97] S. Dandamudi, "The Effect of Scheduling Discipline on Dynamic Load Sharing in Heterogeneous Distributed Systems," *IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS),* Haifa, Israel, January 1997.

[Dan96] S. Dandamudi and H. Hadavi, “Performance Impact of I/O on Sender-Initiated and Receiver-Initiated Load Sharing Policies in Distributed Systems,” *Int. Conf. Parallel and Distributed Computing and Systems,* Dijon, France, 1996, pp. 507-515.

[Dik89] 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.

[Eag86a] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. Software Eng.,* Vol. SE-12, No. 5, May 1986, pp. 662-675.

[Eag86b] 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.

[Eag88] 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.

[Kru88] P. Krueger and M. Livny, "A Comparison of Preemptive and Non-Preemptive Load Distributing," *IEEE Int. Conf. Dist. Computing Systems*, 1988, pp. 123-130.

[Mut91] M. Mutka and M. Livny, "The Availability Capacity of a Privately Owned Workstation Environment," *Performance Evaluation*, Vol. 12, No. 4, July 1991, pp. 269-284.

[Kun91] T. Kunz, "The Influence of Different Workload Descriptions on a Heuristic Load Balancing Scheme," *IEEE Trans. Software Eng.*, Vol. SE-17, No. 7, July 1991, pp. 725-730.

[Mir90] 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.

[Shi92] N. G. Shivaratri, P. Krueger, and M. Singhal, “Load Distributing for Locally Distributed Systems,” *IEEE Computer*, December 1992, pp. 33-44.

[Zho88] S. Zhou, "A Trace-Driven Simulation Study of Dynamic Load Balancing," *IEEE Trans. Software Eng.*, Vol. 14, No.9, September 1988, pp. 1327-1341.

[Zho93] 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.

Appendix A Simulation Model

In the simulation model, a locally distributed system is represented by a collection of nodes. We model communication delays without modelling the low-level protocol details. An Ethernet-like network with 10 Mbps is assumed. The CPU gives preemptive priority to communication activities (such as sending probe messages) 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. Job transmission time is assumed to be uniformly distributed between U_{jx} and L_{jx}.

The system workload is represented by four parameters. The job arrival process at each node is characterized by a mean inter-arrival time $1/\lambda$ and a coefficient of variation (CV) C_a. Jobs are characterized by a processor service demand (with mean $1/\mu$) and a coefficient of variation C_s. We report performance sensitivity to variance in both inter-arrival times and service times (the CV values are varied from 0 to 4). We use exponential distribution to model distributions with a CV of 1 and a two-phase hyperexponential distribution for distributions with CV values from 2 to 4.

Unless otherwise specified, the following default parameter values are used. The distributed system is assumed to have 32 nodes. The average job service time is one time unit. 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. Note that transferring active jobs would incur substantial overhead as the system state would also have to be transferred. Furthermore, it has been shown in [Dan95] that the transfer cost does not change the relative performance of the policies. 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). Similar threshold values are used in previous studies [Eag86a/b, Dik89]. Probe limit P_l is 3 for all policies. For the RR policies, quantum size Q is 0.1 time unit and the context switch overhead is 1% of the quantum size Q. In practice, these values are reasonable.

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%). For the sake of clarity, we have not included the confidence interval information on the plots.

Validation is an important aspect of any simulation-based study. We used three distinct steps to thoroughly validate our simulation model. First, to make sure the various random numbers are properly generated to match the input parameter specifications, we computed the mean and coefficient of variations generated by the simulation model for parameters like the mean and CV of the inter-arrival times of jobs, and mean and CV of the job service times. Second, the parameter values are set such that the the simulation model represents a queueing model whose performance can be obtained analytically. For example, performance of the sender-initiated policy with the FCFS node scheduling policy reduces to no load sharing when the threshold is very high. In this case, the M/M/1 queuing model can be used to verify the results of the simulation experiments. Finally, wherever possible, we verified our results with previously published results.

Appendix B Example Packages

A large number of software packages are available to effectively distribute load among the workstations in a cluster. Such software, referred to as the cluster management software in [Bak96], is available from commercial as well as research sources. A comprehensive review of seven commercial packages and twelve research packages is given in [Bak96]. Details about other systems that are not mentioned in [Bak96] like the V-system, Sprite, and Stealth are given in [Shi92].

Commercial Packages

We describe four commercial packages in this section. All four systems support sequential as well as parallel jobs. However, only the first three systems support process migration and checkpointing.

Codine: Codine (COmputing in DIstributed Network Environments) version 4.0 is marketed by GENIAS Software GmbH, Germany. Codine uses a master daemon, which runs on the main server, to manage the entire cluster. The master daemon receives jobs as well as system state information. Codine allows the users to specify resource requirement when submitting a job. All submitted jobs wait in a job pending queue on the main server. The allocation of jobs in the pending queue to the most suitable node is done by a scheduling daemon. Fault tolerance is provided by maintaining a shadow master daemon, which will transparently spawn another master daemon in case of the master daemon failure. Codine supports a variety of platforms including DEC, HP, IBM, SGI, and Sun. For more details, consult `http://www.genias.de`.

LoadLeveler: LoadLeveler from IBM supports batch job scheduling and uses a Central Manager machine for each cluster. The main responsibilities of the Central Manager are (i) to coordinate LoadLeveler related activities on all machines in a cluster, (ii) to maintain state information on all machines and jobs, and (iii) to assign and schedule jobs to nodes in a cluster. If the Central Manager machine goes down, job scheduling will start when the Central Manager is restarted. In the meantime, users can continue to submit jobs from other machines. LoadLeveler supports a variety of platforms including IBM, HP, SGI and Sun. More information on LoadLeveler can be obtained from

`http://www.rs6000.ibm.com/software/sp_products/loadlev.html`

LSF: LSF (Load Sharing Facility) from Platform Computing supports load sharing within a cluster as well as among several clusters. LSF is based on the Utopia load sharing package developed at the University of Toronto. LSF maintains a master Load Information Manager (LIM) and a slave LIM on each machine in a cluster. The slave LIMs report load information to the master LIM, which performs load distribution within a cluster by using cluster-wide state information. LSF also supports global load sharing among multiple clusters. Multiclusters are treated as multiple autonomous and cooperating load sharing clusters. Within a cluster, the centralized scheduler is used to distribute jobs to appropriate machines. Multicluster load sharing requires cooperation among the master LIMs and is transparent to the user. Systems supported include DEC, IBM, HP, SGI, and Sun. Additional information on LSF can be obtained from `http://www.platform.com/products`.

NQE: NQE (Network Queueing Environment) from Cray Research also uses a central scheduler as in the other three packages. It uses a Central Job Database to hold all pending jobs. When a job is scheduled on a machine, only a copy of it is forwarded to the machine leaving the original in the job database. The main reason for maintaining the original copy at the central job database is to allow for re-execution in case of failure of the assigned machine. Such a mechanism is necessary

as NQE does not support process migration and checkpointing. Systems supported include Cray, DEC, IBM, HP, SGI, and Sun. More information can be obtained from

`http://www.cray.com/products/software/nqe.`

Research Packages

We briefly describe two research packages here. Other popular research packages like the DQS, PBS, and GNQS are reviewed in [Bak96]. Both systems support process migration and checkpointing.

Condor: Condor from the University of Wisconsin uses a combination of centralized and distributed strategies to facilitate load distribution [Lit88]. Each workstation has a local scheduler and a job queue. Jobs submitted at a workstations wait in its local job queue. Each workstation is responsible for scheduling its own jobs. There is, however, a designated centralized controller for load distribution. The controller periodically (at 2-minute intervals) polls each workstation in the cluster to collect state information (their load and whether there are job in its local job queue). The controller uses this state information to assign capacities to workstations, which they use to schedule their local jobs. Spare capacity assignment and hence load distribution requires the central controller; therefore, failure of the controller stops load distribution until it is started on another workstation. Condor supports only sequential jobs - parallel job support is planned for the future. Condor is available in public domain and supports a variety of systems including DEC, IBM, HP, Sun, and Intel.

Batrun: Batrun (Batch After Twilight RUNning) from the Iowa State University also supports only sequential jobs like the Condor [Tan96]. Batrun divides the cluster of workstations into cells. Each cell consists of a cell manager and a number of slave workstations. Each workstation maintains a local queue of jobs as in Condor. The cell manager maintains state information on all the slave workstations within the cell. Cell manager uses this information to allocate idle workstations. If there is a need, it requests idle workstations form another cell. Since there is no central controller, Batrun is robust to workstation failures.

References

[Bak96] M. A. Baker, G. C. Fox, and H. W. Yau, "Review of Cluster management Software," *NHSC Review*, 1996 Volume, First Issue, July 1996 (This paper can be obtained from the URL: `http://www.crpc.rice.edu/NHSEreview`).

[Shi92] N. G. Shivaratri, P. Krueger, and M. Singhal, “Load Distributing for Locally Distributed Systems,” *IEEE Computer,* December 1992, pp. 33-44.

[Tan96] F. Tandiary, S. C. Kothari, A. Dixit, and E. W. Anderson, "Batrun: Utilizing Idle Workstations for Large-Scale Computing," *IEEE Parallel & Distributed Technology*, Summer 1996, pp. 41-48.

[Lit88] M. J. Litzkow, M. Livny, and M. W. Mutka, "Condor - A Hunter of Idle Workstations," *IEEE Int. Conf. Dist. Computing Systems,* San Jose, 1988, pp. 104-111.