Appears in Proc. Int. Conf. on Computers and Their Applications, Honolulu, Hawaii, March 1998.

A Comparative Study of Adaptive and Hierarchical Load Sharing Policies for Distributed Systems

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

Michael Kwok Cheong Lo
Nortel Technologies
Ottawa, Ontario
Canada

TECHNICAL REPORT TR-98-02

ABSTRACT – Dynamic load sharing policies take system state into account in making load distribution decisions. The state information can be maintained in one of two basic ways: distributed or centralized. Two principal types of policies that belong to the distributed scheme are the sender-initiated and receiver-initiated policies. In the centralized scheme, a central coordinator node is responsible for collecting system state information. Distributed policies do not perform as well as the centralized policy. Performance of distributed policies is sensitive to variance in job service times and inter-arrival times. Distributed policies, however, are scalable whereas the centralized policy can cause bottleneck and fault-tolerance problems for large systems. An adaptive distributed policy has been proposed that dynamically switches between sender-initiated and receiver-initiated policies depending on the system state. Here we propose a new global hierarchical load sharing policy that minimizes the drawbacks associated with the distributed and centralized policies while retaining their advantages. We provide a performance comparison of these policies and show that the proposed hierarchical policy provides the best performance among the distributed and adaptive policies for all the various system and workload parameters considered.

1. INTRODUCTION

A large number of software packages are available to effectively distribute load among workstations in a cluster. Such software, referred to as the cluster management software in [1], is available from commercial as well as research sources. A comprehensive review of seven commercial packages and twelve research packages is given in [1].

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 [12]. The chief advantage of the static policies is their simplicity but these policies cannot respond to changes in system state; therefore the performance improvement is limited.

Dynamic policies, on the other hand, use the current or recent system state information in making load distribution decisions. These policies react to system state changes dynamically and therefore there is scope for substantial performance improvements over and above the improvements provided by the static policies. Two types 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 in Section 2.

The final category of adaptive policies change the load sharing policy and/or parameters of the policy in effect depending on the system and workload conditions. Typically adaptive policies use a repertoire of two dynamic policies: a sender-initiated policy and a receiver-initiated policy. For example, at low to moderate system load, a sender-initiated policy might be used; at high system loads, the load sharing policy is switched to a receiver-initiated policy. We describe an adaptive policy in Section 2.

Two important components of a load sharing policy are a transfer policy and a location policy. 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. Load sharing policies can employ either a centralized or a distributed location policy. In a centralized policy, state information is collected by a single coordinator node and other nodes would have to consult this node for advice on the system state in locating a target node. In a distributed policy, system state information is distributed across the nodes in the system. In this type of policy, in order to locate a target node, individual nodes would have to gather the required state information from other nodes. Centralized policy has the advantage of providing near perfect load sharing as the coordinator has the entire system state to make a load distribution decision. The obvious disadvantages of this type of policy are diminished fault-tolerance and potential for performance bottleneck.

The distributed policy eliminates these disadvantages associated with the centralized policy; however, distributed policies may cause performance problems as the state information may have to be transmitted to all nodes in the system. Previous studies have shown that this overhead can be substantially reduced by sampling only a few randomly selected nodes [7]. A further problem with the distributed policies is that the performance of such policies is sensitive to variance in service times as well as inter-arrival times [2, 4]. Distributed policies are, however, scalable to large system sizes.

To overcome these problems, we have proposed a local hierarchical load sharing policy that combines the merits of the centralized and distributed policies while eliminating/minimizing the disadvantages of these policies [5]. In this paper, we propose a modified hierarchical load sharing policy (we call it the *global hierarchical* policy) and focus on the performance of this policy in homogeneous distributed systems. Performance of this policy in heterogeneous distributed systems has been reported in [11]. The results reported here and in [11] suggest that the hierarchical policy provides the best performance among the four non-centralized load sharing policies considered. Its performance is closer to that of the centralized policy, which provides the best performance in the absence of contention.

The remainder of the paper is organized as follows. Section 2 discusses the four load sharing policies against which the performance of the proposed hierarchical policy is compared. The hierarchical policy is described in Section 3. The next section describes the workload and system models and the results are discussed in Section 5. Conclusions are given in Section 6.

2. LOAD SHARING POLICIES

The load sharing policies discussed here use the following transfer policy. When a new job arrives at a node, the transfer policy looks at the job queue length at the node. This queue length includes the jobs waiting to be executed and the job currently being executed. The new job is transferred for local execution if the queue length is less than the specified threshold value T. Otherwise, the job is eligible for a possible remote execution and is placed in the job transfer queue. The location policy, when invoked, will actually perform the node assignment.

2.1. 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 transfer queue. If the job is placed in the job transfer queue, the job is eligible for transfer and the location policy is invoked. The location policy probes (up to) a maximum of probe limit P_l randomly selected nodes to locate a node with the job queue length less than T. If such a node is found, the job is transferred to that node 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 P_l probes fail to locate a suitable node, the job is moved to the job queue to be processed locally. When a transferred job arrives at the destination node, the node must accept and process the transferred job even if the state of the node at that instance has changed since the probing.

2.2. 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 transfer queue of node R as described before. The location policy is typically invoked by nodes at times of job completions. The location policy of node R attempts to transfer a job from its job transfer queue to its job queue if the job transfer queue is not empty. Otherwise, if the job queue length of node R is less than T, it initiates the probing process as in the sender-initiated policy to locate a node S with a non-empty job transfer queue. If such a node is found within P_l probes, a job from the job transfer queue of node S will be transferred to the job queue of node R. In this paper, as in the previous literature [3,7,12], we assume that $T = 1$. 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. Thus the probability of the existence of an idle workstation is high.

Previous implementations of this policy have assumed that, if all probes fail to locate a suitable node to get work from, the node waits for the arrival of a local job. Thus, job transfers are initiated at most once every time a job is completed. This causes 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) [12] or if there is a high variance in job inter-arrival times [2,4]. For example, if four jobs arrive at a node almost at the same time, then this node attempts load distribution only once after completing all four jobs. Worse still is the fact that if there are long gaps in job arrivals, the frequency of load distribution will be low. This adverse impact on performance can be remedied by reinitiating load distribution after the elapse of a predetermined time if the node is still idle. We assume that the receiver-initiated policy uses such a reinitiation strategy.

2.3. Centralized Policy

In this policy, there is a single node (called the "coordinator") that is responsible for collecting the system state information. Whenever that state of a node changes, it informs this change in state to the coordinator for updating purposes. A node can be in one of three states:

- A node is in the *receiver* state if the job queue length of the node is less than T_l (low threshold);
- A node is in the *sender* state if the job queue length of the node is greater than T_h (high threshold);
- otherwise, it is in the OK state.

where $T_h \geq T_l$. In this paper, for reasons explained in Section 2.2, we assume that $T_h = T_l = T = 1$.

The load distribution is initiated by a receiver node (i.e., a node that is in the receiver state). Typically, at job completion times, if the state of the node changes to receiver, it consults the coordinator node for a node that is in the sender state. If a sender node is found, the coordinator informs the sender node to transfer a job to the receiver node. As in the receiver-initiated policy, reinitiation of load distribution is necessary in order to improve its performance under certain system and workload conditions.

2.5. Adaptive Load Sharing Policy

The adaptive policy discussed in [12] is a hybrid policy that has both sender-initiated and receiver-initiated components. It is also an adaptive policy in the sense that it does not select random nodes for probing (as in the sender-initiated and receiver-initiated policies). Instead, each node gathers state information from previous probes and uses this information to select a most likely target node. The state information is gathered at each node by maintaining three lists: senders list, receivers list, and an OK list. Initially, all nodes are considered as receivers. This default assumption is corrected by state information gathered by probing. Thus, at every node, the receivers list is initialized with all nodes except the node itself and the other two lists are empty. At each node i, a node other than i may belong to only one of the three lists, representing the perceived state of that node by node i. Membership of a node changes from one list to another in response to the information received by probing. We now describe how this is accomplished for the sender-initiated and receiver-initiated components. The sender-initiated component is invoked when a node becomes a sender. When a node becomes a receiver, the receiver-initiated component is invoked.

Sender-initiated component: When a node (say node S) becomes a sender, it probes the node (say node R) at the head of the receiver list. When the probe message is received by R, it updates the state information for S to sender and replies with its state information. When S receives the reply, it transfers a job to R if R is a receiver; otherwise, R is moved from its current list to either the sender or OK list depending on the reply.

Receiver-initiated component: When a node (say R) becomes a receiver, it probes a node (say node S) that is selected by first scanning the senders list, then the OK list, and finally the receivers list. The senders list is scanned head to tail as we want to use the up-to-date information on possible sender nodes. The other two lists are scanned tail to head in the hope that using the most out-of-date information, the state might have changed to the sender. We use the receiver list only if the sender and OK lists are empty as a node previously in OK state will have a better chance of becoming a sender than the one that was in the receiver state.

When the probe message is received by S, if it is a sender, it transfer a job to R and informs R of its state after the job transfer. If S is not a sender, it moves R to the head of the receiver list and informs R whether S is in receiver or OK state. On receiving this information from S, R moves S to the head of either receiver or OK list, depending the state of R.

3. Global Hierarchical Load Sharing Policy

There is a tradeoff between the distributed and centralized load sharing policies. While distributed policies are scalable to large systems, performance of these policies is sensitive to variance in service and inter-arrival times. Centralized policy provides near perfect load sharing but causes scalability and fault-tolerance problems. Ideally, we would like to have a policy that provides performance close that of the centralized policy while inheriting the merits of the distributed policies. A local hierarchical load sharing policy has been proposed to achieve this objective [5,10]. Here we propose a new global hierarchical load sharing policy that improves the performance the local policy.

In the proposed global hierarchical policy, instead of a single node maintaining the entire system state, a set of nodes is given this responsibility. The system is *logically* divided into clusters and each cluster of nodes will have a single node that maintains the state information of the nodes within the cluster. The state information on the whole system is maintained in the form of a tree where each tree node maintains the state information on the set of processor nodes in the sub-tree rooted by the tree node. The state information maintained in tree nodes depends on the type of load sharing policy implemented. Figure 3.1 shows an example hierarchical organization for 8 processor nodes with a branching factor *B* of 2. The state information maintained in the hierarchy is useful in implementing the receiver-initiated type of policy.

As shown in Figure 3.1 the hierarchy maintains state information about two states only: the sender state is indicated by a 1 and the non-sender state (receiver or OK) is represented by a 0. Thus, the semantics of the state information kept in the tree nodes are: if there is at least one sender in that branch, a 1 is stored to indicate this fact; otherwise a 0 is stored. For example, the first value in Q2 is 1, which represents the fact that the left sub-tree has at least one sender.

We now describe how the hierarchy-based receiver-initiated policy works. For reasons explained in Section 2.2, we assume a threshold *T* of 1. That is, whenever a node idle, it consults the hierarchy (explained next) for a sender node from which the receiver can get a job. When a job is completed at a node, if there are no jobs in its local job queue, it sends a message to its parent tree queue (i.e., to the processor node that is responsible for maintaining this tree node) for a sender node. If the tree node does not have any sender nodes in its sphere, it forwards the message to its parent tree node. This process is repeated up the tree until either a tree node with a sender node is encountered or the root node is reached. If the root node does not have a sender node implying that there is no sender node in the whole system, a "no job" message is sent back to the receiver node that initiated the request. If a tree node with at least one sender branch is found, the message follows a "downward" path along this branch until it reaches the sender node. If the tree node has more than one sender branch, one is selected at random.

When the message reaches the sender node, if that node is still a sender, it transfers a job to the receiver node. If, on the other hand, the node is no longer a sender as it might be in the process of updating its entry in the hierarchy (we refer to this as the false sender scenario), the search process continues by back tracking until a "true sender" is found[1].

When a "no job" message is received by the receiver node and if the node is still in the receiver state (i.e., there were no local job arrivals since the request) it updates it entry in the hierarchy (to receiver state) and waits for the corresponding reinitiation period and initiates another load distribution request (if it remains in the receiver state at the end of the reinitiation period).

Since load sharing can be done at various levels of the tree, the root node does not have to handle requests form all nodes in the system. Furthermore, it can also be seen that the set of nodes that form the tree can all be distributed to different system nodes so that no node is unduly overloaded with maintenance of system state information. For a system with *N* nodes, the maximum number of tree nodes (*N*-1) will be required when the branching factor *B* is 2. Since there are *N* system nodes these (*N*-1) tree nodes can be distributed uniformly across the nodes in the system.

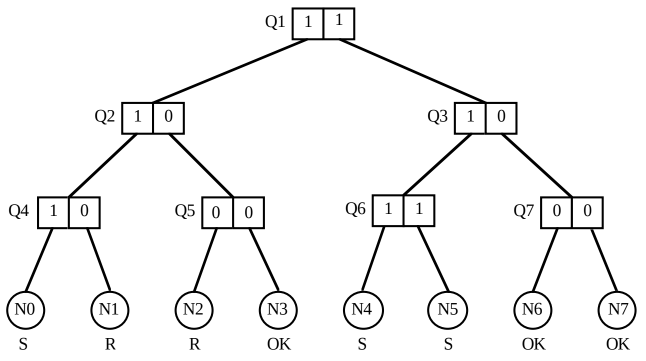

Figure 3.1 An example hierarchical organization for an 8-node system with a branching factor of 2 (S = sender, R = receiver, OK = normal load)

4. System and Workload Models

In the simulation model, a locally distributed system is represented by a collection of nodes. In this paper we consider only homogeneous nodes. We also 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 is 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 in [6] uses serial probing.

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 C_a. Jobs are characterized by a processor service demand (with mean $1/\mu$) and a coefficient of variation C_s. 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 have used a two-stage hyperexponential model to generate service times and inter-arrival times with CVs greater than 1.

5. Performance Analysis

This section presents the simulation results and discusses the performance sensitivity of the four load sharing policies. Unless otherwise specified, the following default parameter values are assumed. The distributed system is assumed to have N = 32 nodes interconnected by a 10 megabit/second communication network. The average job service time $1/\mu$ is one time unit. 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. The load distribution reinitiation period when a "no job" message is received is fixed at 1. 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

[1] Unless there is no sender node in the entire system; in this case, the root tree node sends a "no job" message to the originator of the request.

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. A probe limit P_l of 3 is used for the sender-initiated and receiver-initiated policies. For the hierarchical policy, the default branching factor is fixed at 8. Expanded version of this paper discusses the impact of this parameter on the performance of the hierarchical policy.

A batch strategy has been used to compute confidence intervals (at least 30 batch runs were used for the results reported here). This strategy has 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.

Performance comparison of the local and global hierarchical policies in [9] indicates that the global policy performs better than the local policy. For the sake of brevity, we include only a brief performance comparison of the local and global hierarchical policies. Further, we do not include the performance of the sender-initiated and receiver-initiated polices as they perform substantially worse than the other three policies [2]. Thus, in the following discussion, we compare the performance of the global hierarchical policy against the adaptive and centralized policies.

5.1. Basic Performance Comparison

Figure 5.1 shows the mean response time of the three load sharing policies as a function of offered system load. Note that the offered system load is given by λ/μ. Since $\mu = 1$ for all the experiments, offered system load is equal to λ. These results correspond to an inter-arrival CV (C_a) and service time CV (C_s) of 4. These CV values are reasonable and supported by empirical data. For example, the Berkeley data collected by Zhou [13] indicates that the coefficient of variation of inter-arrival times is 2.65 and that of service times is 12.83! Due to lack of space, we have not included the results for other values of inter-arrival and service CVs. However, the following sections discuss the performance sensitivity to inter-arrival and service time variances.

The data in Figure 5.1 suggest that the adaptive policy performs marginally better than the hierarchical policy for low to moderate system loads. For example, performance of the adaptive policy is better than the hierarchical policy up to an offered system load of about 50%. In fact, at very low system loads (for example, at load 40% in Figure 5.1), the performance of the adaptive policy is even better (though marginally) than the centralized policy. This is because, the centralized policy is implemented as a receiver-initiated policy as opposed to a symmetric policy (as in the adaptive policy) that incorporates both sender-initiated and receiver-initiated policy. Since our interest is in moderate to high system loads, we have not focussed on this minor aspect of the policies in this study.

The performance of the hierarchical policy is better than the adaptive policy at moderate to high system loads. The performance of the adaptive policy deteriorates rapidly relative to the centralized policy as the system load increases. On the other hand, the hierarchical policy provides performance very close to that of the centralized policy even at high system loads. For example, the relative performance deterioration of the adaptive policy is approximately 13% and that of the hierarchical policy is about 3.8% when the offered system load is 80%. The corresponding values, when the load is 90%, are 20% for the adaptive policy and 6% for the hierarchical policy. The main reason for this is that the adaptive policy is a distributed policy and it is sensitive to clustered arrival of jobs (i.e., high inter-arrival variance) as discussed in the next section.

5.2. Sensitivity to Variance in Inter-Arrival Times

The effect of inter-arrival CV C_a is shown in Figure 5.2. The service time CV is set to 1. These results show that the adaptive policy exhibits more sensitivity to the variance in inter-arrival times. For example, when the inter-arrival CV is 2, the performance of the adaptive policy is worse by 9.2% relative to that of the centralized policy; the corresponding value for the hierarchical policy is 2.2%. These value increase to 16% and 3%, respectively, when the inter-arrival CV is increases to 4. The main reason for these differences is that the hierarchical policy is able to reflect the system state changes much more quickly than the adaptive policy. Since higher inter-arrival CV implies clustered nature of job arrivals into the system, delayed reflection of system state results in performance deterioration.

5.3. Sensitivity to Variance in Service Times

Next we look at the the impact of the service time CV on the performance of these three policies. In this context, it should be noted that the service time distribution of the data collected by Leland and Ott [8] from over 9.5 million processes has been shown to have a coefficient of variation of 5.3 and the data collected by Zhou at Berkeley [13] indicates a service time CV of 12.83. To study the impact of service time CV, we vary the CV value from 1 to 4.

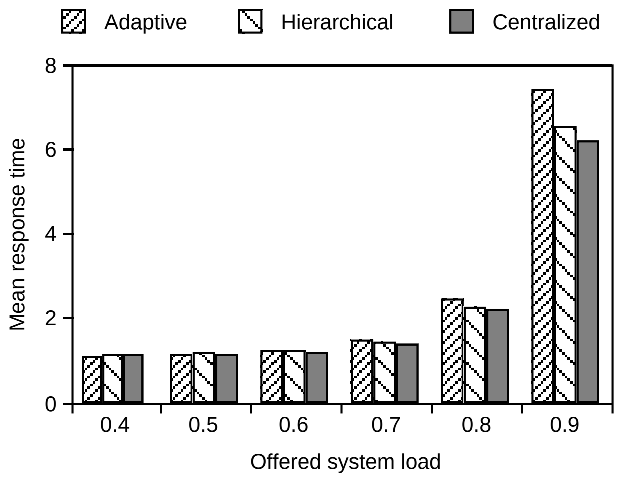

Figure 5.1 Performance sensitivity as a function of offered system load (N = 32 nodes, $\mu = 1$, $C_s = 4$, λ is varied to vary system load, $C_a = 4$, $B = 8$, $T = 1$, $P_l = 3$, transfer cost = 1%)

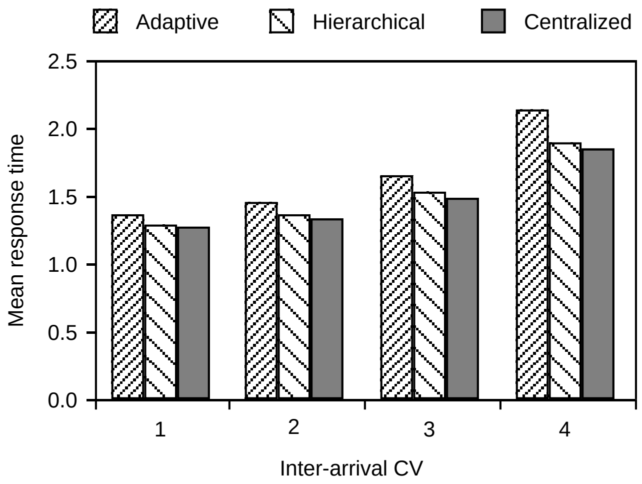

Figure 5.2 Performance sensitivity to inter-arrival time CV (N = 32 nodes, $\lambda = 0.8$, C_a is varied from 0 to 4, $\mu = 1$, $C_s = 1$, $B = 8$, $T = 1$, $P_l = 3$, transfer cost = 1%)

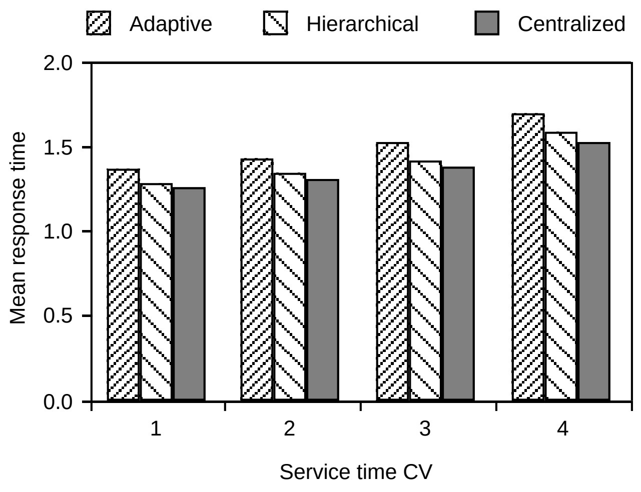

Figure 5.3 Performance sensitivity to service time CV (N = 32 nodes, λ = 0.8, C_a = 1, μ = 1, C_s = varied from 0 to 4, B = 8, T = 1, P_l = 3, transfer cost = 1%)

Figure 5.3 shows the impact of variance in service time. For this experiment, the inter-arrival time CV is fixed at 1. All three policies exhibit less sensitivity to service time variance than to inter-arrival time variance. As demonstrated in [2, 4], clustered arrival jobs (the result of high inter-arrival CV) causes the sender-initiated policy to perform better load distribution than the receiver-initiated policy. On the other hand, the receiver-initiated policy can perform better load distribution when the service time CV is high. For the system load under consideration, all the policies policies use the receiver-initiated component. Thus, these policies exhibit more sensitivity to inter-arrival time CV than to service time CV.

The performance deterioration of the adaptive and hierarchical policies increases slowly with service time CV. For example, the relative performance of the adaptive policy is worse by 9% when the service CV is 2; the corresponding value of the hierarchical policy is 2.4%. These values increase only to 10.6% and 3.4%, respectively, when the service time CV is increased to 4.

5.4. Sensitivity to the Branching Factor

Performance of the hierarchical policy is sensitive to the branching factor B of the hierarchy. The results presented in previous sections have used a branching factor of 8. This section discusses the sensitivity of the hierarchical policy to this design parameter. Figure 5.4 presents the results for various branching factors for a system consisting of 32 nodes (with an offered system load of 80%). As in Section 5.1, the CV values of the service and inter-arrival times are both fixed at 4.

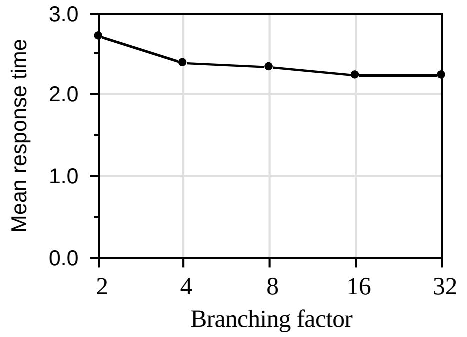

Figure 5.4 Sensitivity of the hierarchical policy to the branching factor (N = 32 nodes, λ = 0.8, Ca = 4, μ = 1, Cs = 4, T = 1, transfer cost = 1%)

As can be expected, the performance improves with increasing branching factor. Note that the results for the branching factor of 32 correspond to the single coordinator case. However, the system simulated is fairly small (32 nodes) as it takes large amount of resources for running simulations of large systems. In large systems, the single coordinator may not provide substantial performance improvements.

5.5. Performance of Local and Global Policies

In the local hierarchical policy proposed in [5], the state information maintained in the hierarchy is different from that used in the global policy (shown in Figure 3.1). In the local policy, a more detailed state information is stored as shown in Figure 5.5. Recall that a node can be in one of three states: *sender* (overloaded), *OK* (normal load), or *receiver* (underloaded). We use +1 to represent the sender state, 0 for the OK state, and –1 for the receiver state. For example, Q4 maintains the state information in nodes N0 ands N1. To see the difference between the local and global policies, compare the state information maintained in the hierarchy by the two policies (see Figures 3.1 and 5.5). Unlike the global policy, the local policy maintains the arithmetic sum of the state information of the tree nodes below it. If this sum is positive (negative), we store +1 (–1) to represent that the cluster is in sender (receiver) state; a zero represents that the cluster is in OK state. For example, summary state metric for Q4 is zero, which implies that the cluster represented by Q4 (i.e., nodes N0 and N1) is in OK state. Thus this policy encourages local load balancing. For this reason, this policy is called the *local hierarchical* policy. As discussed next, this behaviour leads to performance deterioration relative to the global policy. Also note that the summary metric for Q6 stored in Q3 is +1 rather than +2. An advantage of this scheme is that it reduces the number of updates required to maintain the hierarchy. For example, if the system state changes and N4 moves to OK state, only the entry in Q6 needs to be changed. Since N5 is still a sender, no state change is necessary for Q3.

The performance of the local and global polices is shown in Figure 5.6. For comparison, we have included the centralized and the adaptive policies as well. The adaptive policy performs marginally better than the local hierarchical policy. The results also show that the local policy performs worse than the global policy for the reasons discussed. However, the local hierarchical policy requires fewer messages for load distribution. To show this we plot the average message handling rate of the tree nodes at each level i. This rate is defined as

$$\frac{\#\text{ of mesages received} + \#\text{ of messages sent out by all nodes at level } i}{\#\text{ of tree nodes at level } i * \#\text{ of jobs completed}}$$

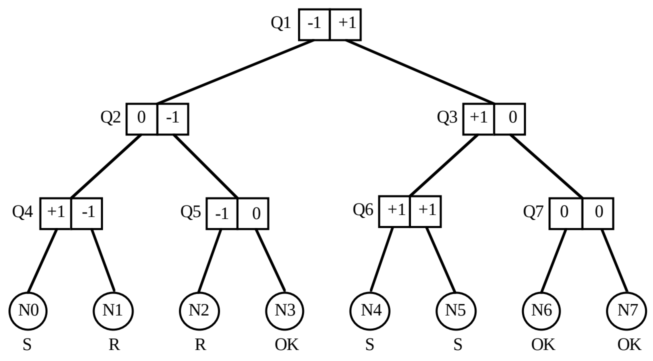

Figure 5.5 An example hierarchical organization for an 8-node system with a branching factor of 2 for the local hierarchical policy (S = sender, R = receiver, OK = normal load)

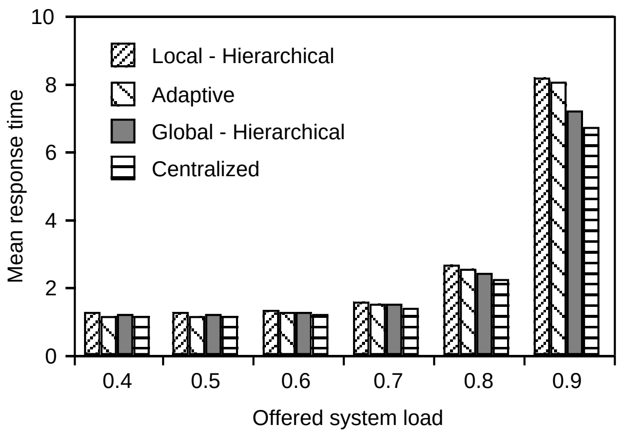

Figure 5.6 Performance sensitivity as a function of offered system load (N = 32 nodes, μ = 1, C_s = 4, λ is varied to vary system load, C_a = 4, B = 4, T = 1, P_l = 3, transfer cost = 1%)

Figure 5.7 shows the message handling rate ratio relative to the message handling rate of the centralized policy when the CV values of service times and inter-arrival times are both 1 and the system load is 80%. The results are for a branching factor of 4. The rate is the highest at the middle level. At this level the local hierarchical policy reduces the message rate by a factor of 2.5. The corresponding value for the global policy is 2. The message handling rate of leaf nodes is low (about 18% - 20% of the centralized rate) because there are a large number of them. For our example, there are 8 leaf level tree nodes whereas only two middle level nodes. The root node in the hierarchical policy handles only about 25% of the centralized rate. The global policy handles about 35%. Similar trends have been observed with other parameter values (details can be found in [9]).

6. Conclusions

There is a tradeoff between distributed and centralized load sharing policies. While the distributed policies are scalable to large systems, the performance of these policies is sensitive to variance in service times and inter-arrival times. Centralized policy provides near perfect load sharing but causes scalability and fault-tolerance problems. Ideally, we would like to have a policy that provides performance very close that of the centralized policy while inheriting the merits of the distributed policies.

Here we have proposed a new global hierarchical load sharing policy that minimizes the drawbacks associated with the distributed and centralized policies while retaining their advantages. In order to see how close the hierarchical policy performs in comparison to the single coordinator policy, we have considered the scenario where the bottleneck problem does not exist in the centralized policy. Our results show that the proposed hierarchical policy provides the best performance among the distributed and adaptive policies and performs very close to that of the centralized policy (which provides the best performance in the absence of contention) for all the various system and workload parameters considered in this study. We have compared performance of these policies in heterogeneous systems as well [9]. These results also indicate that the hierarchical policy provides the best performance.

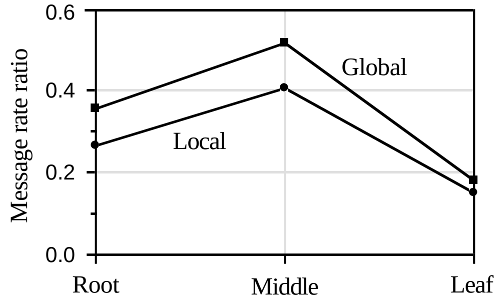

Figure 5.7 Message handling rate ratio (relative to the message handling rate of the centralized policy) of nodes at the three levels of the hierarchy (N = 32 nodes, λ = 0.8, Ca = 1, μ = 1, Cs = 1, T = 1, B = 4, transfer cost = 1%)

Acknowledgements

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

References

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

[2] S. P. Dandamudi, "Sensitivity Evaluation of Dynamic Load Sharing in Distributed Systems," *IEEE Concurrency* (to appear).

[3] S. P. 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, 1997, pp. 17-24.

[4] 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.

[5] 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, pp. 3-10.

[6] 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.

[7] 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.

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

[9] M. Lo, *Performance of Load Sharing Policies in Distributed Systems*, MCS Thesis, School of Computer Science, Carleton University, Ottawa, Canada, 1996.

[10] 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.

[11] M. Lo and S. Dandamudi,"Performance Comparison of Adaptive and Hierarchical Load Sharing in Heterogeneous Distributed Systems," *Int. Conf. on Parallel and Distributed Computing Systems*, New Orleans, 1997, pp. 524-528.

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

[13] 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.