This paper appears in Int. Conf. Parallel and Distributed Computing Systems, New Orleans, 1997.

Performance Comparison of Adaptive and Hierarchical Load Sharing in Heterogeneous Distributed Systems

Michael Kwok Cheong Lo
Nortel Technologies
Ottawa, Ontario
Canada

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

TECHNICAL REPORT TR-98-03

ABSTRACT – State information in dynamic load sharing policies can be maintained in one of two basic ways: distributed or centralized. Two principal types of distributed policies 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. 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 system state. A global hierarchical policy has been proposed to minimize the drawbacks associated with the distributed and centralized policies while retaining their advantages. This paper provides a performance comparison of these policies in heterogeneous distributed systems.

1. INTRODUCTION

Dynamic load sharing policies use the current or recent system state information to make load distribution decisions. These policies react to system state changes dynamically and provide substantial performance improvements over no load sharing. Two principal types of dynamic policies 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 heavily-loaded nodes from which work may be transferred.

Adaptive load sharing policies change the load sharing policy and/or parameters of the policy in effect depending on system and workload conditions. Typically adaptive policies use a repertoire of two dynamic policies: sender-initiated and receiver-initiated dynamic policies. 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 will 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. CPU queue length is found to be the most effective load index [7].

Load sharing policies 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 consult this node for advice on the system state. 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. Centralized policies are suitable for load distribution within a cluster of workstations. A comprehensive review of several commercial load distribution packages (such as Loadleveler from IBM, LSF from Platform Computing, Codine from Genias Software, and NQE from Cray Research) and research packages (such as Condor) is given in [1].

Distributed policies eliminate the 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 [6]. 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, 5]. Distributed policies are, however, scalable to large system sizes.

To overcome these problems, we have proposed local and global hierarchical load sharing policies that combine the merits of the centralized and distributed policies while minimizing the disadvantages of these policies [3, 8, 9]. In this paper, we focus on the performance of the adaptive and global hierarchical policies in heterogeneous distributed systems. The results reported here suggest that both policies perform well in heterogeneous systems. They both provide performance close to the centralized policy. The hierarchical policy provides better performance than the adaptive policy under certain conditions.

Location policies can be divided into two basic classes: sender-initiated or receiver-initiated. It has been shown that, when the first-come/first-served scheduling policy is used, sender-initiated policies perform better than receiver-initiated policies at low to moderate system loads [6]. This is because, at these system loads, the probability of finding a lightly-loaded node is higher than that of finding a heavily-loaded node. At high system loads, on the other hand, receiver-initiated policies are better because it is much easier to find a heavily-loaded node. Adaptive policies have been proposed to incorporate the good features of both these policies [11, 12]. We describe this adaptive policy in Section 2.

Mirchandani et al. [10] consider the impact of delay on the performance of heterogeneous distributed systems. They consider two types of heterogeneous systems: in type I systems external job arrival rates at nodes differ and in type II systems the processing rates of the nodes also differ. As in their study, we consider these two types of heterogeneous systems.

The remainder of the paper is organized as follows. Section 2 discusses the three load sharing policies – centralized, adaptive, and hierarchical policies. Section 3 describes the workload and system models that were used to conduct the simulation experiments. The results for type I and type II heterogeneous systems are discussed in Sections 4 and 5, respectively. Conclusions are given in Section 6.

This report can be obtained from `http://www.scs.carleton.ca`

2. LOAD SHARING POLICIES

This section describes the three load sharing policies: centralized single coordinator, adaptive, and hierarchical policies. We assume the first-come/first-served node scheduling policy. Job transfers are assumed to be done on a non-preemptive basis.

2.1. Single Coordinator Policy

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

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

where $T_h \geq T_l$. In this paper we assume that $T_h = T_l = 1$. Note that $T_l = 1$ implies that 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 an idle workstation existing in a system is high.

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. If no sender is found, the receiver waits for a reinitiation period and initiates another load distribution request (if it remains in the receiver state during the reinitiation period).

2.2. Adaptive Load Sharing Policy

The adaptive policy proposed in [11] has sender-initiated and receiver-initiated components. This policy behaves like the sender-initiated policy at low to moderate system loads and like the receiver-initiated policy at moderate to high system loads. This policy is also adaptive in the sense that it does not select random nodes for probing. Instead, each node gathers state information from the 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*.

2.3. Global Hierarchical Load Sharing Policy

In this 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 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 state information on the set of nodes in the sub-tree rooted by the tree node. The state information maintained in tree nodes depends on the type of location policy used. Figure 1 shows an example hierarchical organization for 8 nodes with a branching factor *B* of 2. The state information given in this figure is useful for implementing the receiver-initiated type of policy.

As shown in Figure 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 Q3 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

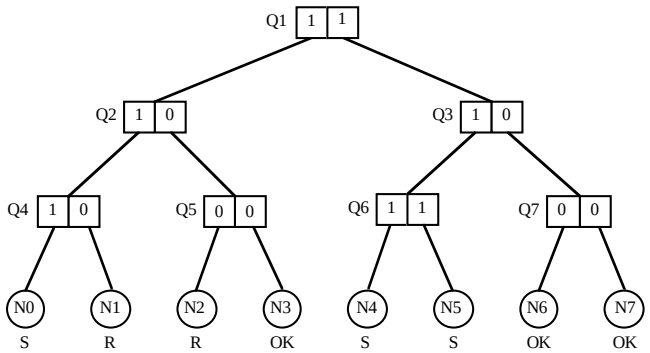

Figure 1 An example hierarchical organization for an 8-node system with a branching factor *B* of 2 (S = sender, R = receiver, OK = normal load)

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 during 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.

3. System and Workload Models

In the simulation model, a locally distributed system is represented by a collection of nodes. We model the communication network in the system at a higher level. We model communication delays without modelling the low-level protocol details. Each node is assumed to have a communication processor that is responsible for handling communication with other nodes. Similar assumptions are made by other researchers [10]. The CPU would give preemptive priority to communication activities (such as sending a probe message) over the processing of jobs.

The CPU overheads to send/receive a probe and to transfer a job are modelled by T_{probe} and T_{jx}, respectively. Actual job transmission (handled by the communication processor) time is assumed to be uniformly distributed between U_{jx} and L_{jx}. Probing is assumed to be done serially, not in parallel.

The system workload is represented by four parameters. The job arrival process at each class i node is characterized by a mean inter-arrival time $1/\lambda_i$ and a coefficient of variation C_{ai}. Jobs are characterized by a processor service demand on a class i node (with mean $1/\mu_i$) and a coefficient of variation C_{si}. We study the performance sensitivity to variance in both inter-arrival times and service times (the CV values are varied from 0 to 4). Sensitivity to service time variance is given in [9]. We have used a two-stage hyperexponential model to generate service times and interarrival times with CVs greater than 1.

As in [10], we consider two types of heterogeneous systems. In type I systems, the nodes are homogeneous (i.e., have the same processing rate) but the job arrival rates at nodes are different. To reduce the complexity of the experiments, we consider two node classes as in [10]. In type II systems, the node processing rates are also different. We consider two node classes in type II systems as well. Nodes in each class in type II system have different job arrival rates and service rates.

As in most previous studies, we model only CPU-intensive jobs in this study. We use the mean response time as the chief performance metric to compare the performance of the three load sharing policies.

[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.

4. Performance of Type I Systems

This section presents the simulation results for type I heterogeneous systems. Unless otherwise stated, the following default parameter values are assumed. The distributed system has $N = 32$ nodes. The number of class 1 nodes N_1 is 6 and the number of class 2 nodes N_2 is 26. Section 4.2 discusses the impact of node distribution between class 1 and class 2. The average job service time is one time unit for both classes. That is $\mu_1 = \mu_2 = 1$. The 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 service time). Since we consider only non-executing jobs for transfer, 1% is a reasonable value. The threshold T is set at 1. Probe limit P_l is 3 for the adaptive policy.

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

4.1. Basic Performance Comparison

Figure 2 shows the mean response time as a function of offered system load on class 2 nodes. Note that the offered system load for class i is given by λ_i/μ_i. Since $\mu_1 = \mu_2 = 1$, offered system load on class i nodes is equal to λ_i. In this experiment we have fixed λ_1 at 0.9 and λ_2 is varied to see the performance impact of system load. The results in Figure 2 correspond to an inter-arrival CV (C_{ai}) and service time CV (C_{si}) of 4 for both classes. 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, Section 4.3 discusses the performance sensitivity to variance in inter-arrival times. The impact of service time variance is given in [9].

The results in Figure 2 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 marginally better than the hierarchical policy up to an offered system load on class 2 nodes of about 50%. In fact, at very low system loads (for example, at load 40% in Figure 2), the performance of the adaptive policy is even better (though

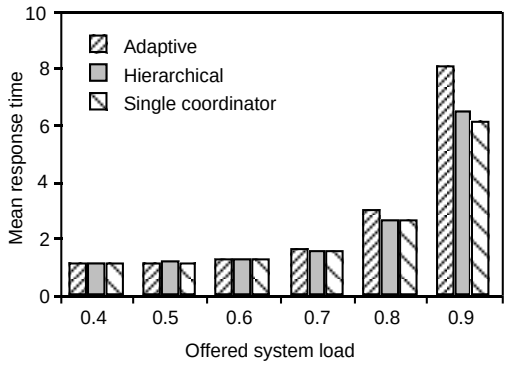

Figure 2 Performance sensitivity to offered system load (N = 32 nodes, $N_1 = 6$, $N_2 = 26$, $\mu_1 = \mu_2 = 1$, $C_{s1} = C_{s2} = 4$, $\lambda_1 = 0.9$, λ_2 is varied to vary class 2 system load, $C_{a1} = C_{a2} = 4$, $B = 8$, $T_l = T_h = 1$ for both classes, $P_l = 3$, transfer cost = 1%)

marginally) than the single coordinator policy. This is because, the single coordinator 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.

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 single coordinator policy as the system load increases. On the other hand, the hierarchical policy provide performance very close to that of the single coordinator policy even at high system loads. For example, the relative performance deterioration of the adaptive policy is approximately 14% and that of the hierarchical policy is about 2% when the offered system load on class 2 nodes is 80%. The corresponding values, when the load is 90%, are 31% 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) and degree of heterogeneity as discussed in the next two sections.

4.2. Impact of degree of heterogeneity

The results presented in the last section assumed that the class 1 nodes $N_1 = 6$ and class 2 nodes $N_2 = 26$, where the 6 class 1 nodes are heavily loaded (offered load of 90%). In this experiment, the offered system loads for class 1 and 2 are fixed as 0.1 and 0.9. Figure 3 shows the results of the experiment. In this graph, the x-axis gives the number of class 2 nodes N_2 and remaining nodes belong to class 1 group. Since class 1 nodes are lightly loaded, the mean response time increases with N_2.

When the number of class 2 nodes N_2 is smaller, the adaptive policy performs better than the other two policies. This is because the system is lightly loaded under these conditions and, for reasons explained in Section 4.1, the adaptive policy performs better. For example, when N_2 is 1, there are 31 nodes operating at 10% offered load and only one node that is at 90% offered system load. This situation is ideal for sender-initiated load sharing. Since the single coordinator and hierarchical policies do not include a sender-initiated component, they perform worse than the adaptive policy. If performance at this load is important, both these policies can be augmented just like the adaptive policy to include a sender-initiated component.

The weakness of the adaptive is apparent at the other end of the spectrum where, for example, N_2 is 31. In this case, there is only one node that is operating at 10% offered load while all the other 31 nodes are at 90% load. The receiver-initiated component is ideal in this scenario in locating a heavily loaded node. For example, when N_2 is 26, the relative performance deterioration (relative to that of the single coordinator) of the adaptive policy is 9.6% and that of the hierarchical policy is 1.5%. The corresponding values increase to 19% and 5.2% when N_2 is 31. The main reason for the performance deterioration of the adaptive policy is that the state information has to be gathered in a distributed manner. Thus, as this activity takes some time, the policy is not able to respond to quickly to the state changes in the system.

The hierarchical policy experiences performance deterioration as it would require more update messages as the system load increases. This is because, as N_2 increases, the system moves to a higher system load state as the class 2 nodes are at 90% offered system load. At these system loads, the additional delay associated with querying the hierarchy for a sender node increases. This additional message delay is a function of the branching factor. Although not shown here, we have conducted experiments with varying branching factors and shown that the performance of the hierarchical policy for practical branching factor values (e.g., equal to a cluster size) is superior to that of the adaptive policy.

4.3. Sensitivity to variance in arrival times

The effect of inter-arrival CV C_{ai} is shown in Figure 4. For both classes, the service time CV is fixed at 1. The inter-arrival time CV is set to the same value for both classes and is shown on the x-axis. 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 6.2% relative to that of the single coordinator policy; the corresponding value for the hierarchical policy is 2.3%. These value increase to 12.3% and 2.8%, 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. Performance of Type II systems

In type II heterogeneous systems, nodes may have different processing rates in addition to seeing different job arrival rates as in type I systems. As in the type I systems, we consider a system with $N = 32$ nodes consisting of two node classes with processing rates of $\mu_1 = 0.5$ and $\mu_2 = 1$. Due to space limitations, we will only present a sample of the simulation results here.

Figure 5 presents the results as a function of offered system load. For this experiment, we have divided the 32 system nodes equally between the two node classes (i.e., $N_1 = N_2 = 16$). The

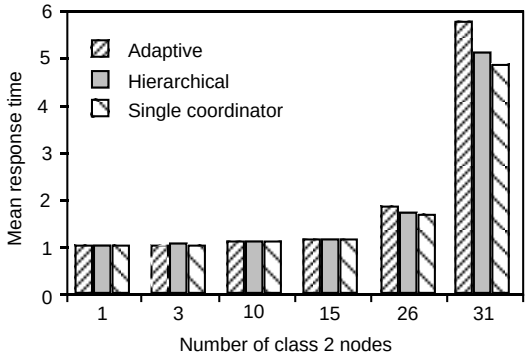

Figure 3 Performance sensitivity to degree of heterogeneity (N = 32 nodes, $N_1 = N - N_2$, N_2 is varied, $\mu_1 = \mu_2 = 1$, $C_{s1} = C_{s2}$ = 4, $\lambda_1 = 0.1$, $\lambda_2 = 0.9$, $C_{a1} = C_{a2} = 4$, $B = 8$, $T_l = T_h = 1$ for both classes, $P_l = 3$, transfer cost = 1%)

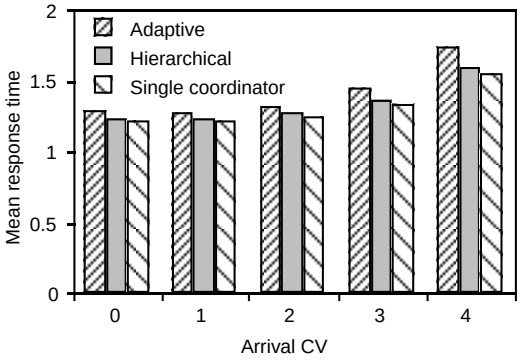

Figure 4 Performance sensitivity to inter-arrival time CV (N = 32 nodes, $N_1 = 6$, $N_2 = 26$, $\lambda_1 = 0.1$, $\lambda_2 = 0.9$, $C_{a1} = C_{a2}$ = varied from 0 to 4, $\mu_1 = \mu_2 = 1$, $C_{s1} = C_{s2} = 1$, $B = 8$, $T_l = T_h = 1$ for both classes, $P_l = 3$, transfer cost = 1%)

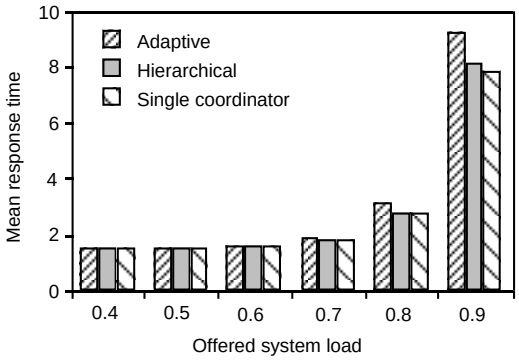

Figure 5 Performance sensitivity to offered system load (N = 32 nodes, $N_1 = N_2 = 16$, $\lambda_1 = \lambda_2/2$, λ_2 is varied, $C_{a1} = C_{a2} = 4$, $\mu_1 = 0.5$, $\mu_2 = 1$, $C_{s1} = C_{s2} = 4$, $B = 8$, $T_l = T_h = 1$ for both classes, P_l =3, transfer cost = 1%)

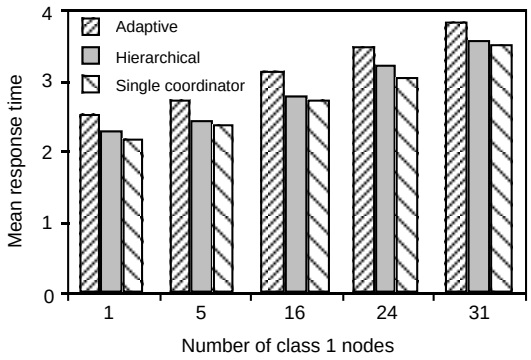

Figure 6 Performance sensitivity to degree of heterogeneity (N = 32 nodes, N_1 is varied, $N_2 = N - N_1$, $\lambda_1 = 0.4$, $\lambda_2 = 0.8$, $C_{a1} = C_{a2} = 4$, $\mu_1 = 0.5$, $\mu_2 = 1$, $C_{S1} = C_{S2} = 4$, $B = 8$, $T_l = T_h = 1$ for both classes, P_l =3, transfer cost = 1%)

offered system load is maintained the same for both classes. That is, the arrival rate of class 1 is maintained at half of that for class 2 nodes because class 2 nodes are twice as fast (i.e., $\lambda_1 = \lambda_2/2$). The theshold values T_l and T_h are fixed at 1 for both classes. The inter-arrival and service time CVs for both classes are fixed at 4 .

All policies exhibit similar behaviour as in Figure 2. The mean response times in Figure 4 are higher than those in Figure 2 as the system is operating at a higher average system load. The performance of the adaptive and hierarchical policies is worse by 14% and 1%, respectively, when the offered system load is 80%; the corresponding values are 17.8% and 3.8%, respectively, when the system load is increased to 90%. The reduced performance sensitivity is due to the fact that each class has 16 nodes as opposed 6 and 26 in type I experiment.

We next look at the impact of degree of heterogeneity on the system performance. Figure 6 shows the response time as a function of number of class 1 nodes N_1. Note that the number of class 2 nodes (i.e., the faster nodes) N_2 is given by $32–N_1$. Thus as we move from left to right in Figure 6, the number of slower nodes increases in the system. As a result all policies experience an increase in response time. However, because each class of nodes is operating at the same system load of 80%, the performance sensitivity is much smaller than that shown in Figure 3 for the type I workload. Except for this difference, the policies exhibit similar behaviour as in Figure 3.

6. CONCLUSIONS

We have provided a performance comparison of adaptive and global hierarchical load sharing policies in heterogeneous distributed systems. In order to see how close the adaptive and hierarchical policies perform in comparison to the centralized single coordinator policy, we have considered the scenario where the bottleneck problem does not exist in the centralized policy.

Our results show that the adaptive policy performs well except in certain cases. As the adaptive policy is a distributed policy, it suffers at high systems loads, high variance in job inter-arrival times (i.e., clustered arrival) and high degree of heterogeneity. The global hierarchical policy provides better performance than the adaptive policy and its performance is very close to that of the single coordinator policy (which provides the best performance in the absence of contention) for all the various system and workload parameters considered in this study.

ACKNOWLEDGEMENTS

We gratefully acknowledge the financial support provided by the Natural Sciences and Engineering Research Council of Canada and Carleton University. This work was done while the first author was at the School of Computer Science, 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 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.

[4] 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, pp. 17-24.

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

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

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

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

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

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

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

[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.