1991

A decentralized task scheduling algorithm and its performance modeling for computer networks

Ishfaq Ahmad
Syracuse University

Arif Ghafoor
Purdue University

Kishan Mehrotra
Syracuse University, mehrotra@syr.edu

A Decentralized Task Scheduling Algorithm and its Performance Modeling for Computer Networks

Ishfaq Ahmad, Arif Ghafoor *, and Kishan Mehrotra

School of Computer and Information Science, Syracuse University, Syracuse, NY 13244

* School of Electrical Engineering, Purdue University, West Lafayette, IN 47907

Abstract

A dynamic task scheduling algorithm, that is stable, decentralized, and adaptive to network topology, is presented. The proposed algorithm is an extension of nearest neighbor load balancing strategy with an enhanced degree of efficiency and it is intended for multicomputers connected by a store and forward communication network. The proposed algorithm is modeled by a central server open queuing network. It is shown that the response time of a task consists of two parts. The first part comprises a task's settling time which consists of scheduling time, communication time, and waiting time in scheduling and communication queues. The second part comprises waiting time in the execution queue in the execution time itself. In order to reduce the first response time, the scheduling algorithm needs to be stable, so that a task is quickly settled at some node. On the other hand, the second response time is reduced if the algorithm efficiently migrates the task to a lightly loaded node. The proposed algorithm is comprehensively evaluated, through simulation and analytical model, and is shown to be both stable and efficient. For performance evaluation, the task transfer cost and the scheduling overhead is also taken into account. Experimental results are also obtained for another nearest neighbor scheduling scheme and compared with the proposed algorithm.

1. Introduction

In order to realize the full potential of a multicomputer network, workload scheduling and remote execution facilities must be carefully designed. The scheduling strategies should incur less overhead and identify suitable remote sites for migrating extra workload. This paper presents a decentralized task scheduling algorithm which is based on nearest neighbor load balancing. The performance of the algorithm is modeled by an analytical model as well as simulation. It is shown that if a task is allowed to migrate a number of times, the response time of a task can be analyzed in two phases. The first phase comprises a task's settling time which consists of scheduling and communication times, and waiting times in scheduling and communication queues at various nodes. If the task is scheduled at some node, the second phase comprises waiting time in the execution queue in the execution time itself. In order to reduce the first response time,called task settling time, the scheduling algorithm needs to be stable, so that a task is quickly settled at some node, without making unnecessary migrations.

The settling time of a task can become very high if the system enters into task–thrashing state. Task–thrashing occurs if nodes spend more time on task migration than on task execution [7]. In this situation, a lightly loaded node can be dumped with tasks from heavily loaded nodes. The other problem associated with dynamic scheduling algorithms is state–woggling which is described as a state in which processors frequently change their status between low and high [10]. On the other hand, the second phase of response time is reduced if the algorithm is accurate migrates tasks to a lightly loaded nodes, and achieves a good load balance. Clearly, the performance of any scheduling algorithm depends on reducing both types of response times. In addition, system topology can have a direct effect on the overall performance. It is very likely that a scheduling strategy may perform well on one topology but may perform poorly on the other. Furthermore, the scheduling overhead and communication cost for task migration can not be ignored.

The proposed algorithm is designed to meet these objectives. For analyzing its performance, we use both analytical modeling and simulation. The analytical model, itself, consists of queuing, statistical and simulation experiments. The proposed strategy is modeled by a central server queuing network. For the queuing model, the proposed scheduling strategy is characterized by the probability that a newly arrived task is executed locally or migrated to another node. This probability determines the tendency of task migration which in turn determines affects the settling time of a task. The execution queue length representing the load at a node is also determined. The queue length at a node affects the second part of a task's response time. In our performance evaluation methodology, we obtained a large number of values of the average queue length and the probability associated with task migration, through simulation. We characterize these two parameters through, statistical analysis, in terms of system parameters. For details see [2]. The analytical queuing model uses statistical analysis to find the response time of a system with any set of system parameters. Another nearest neighbor scheduling strategy is characterized by the proposed performance evaluation model and compared with the proposed algorithm. These results are, again, compared with independent simulation. The main advantage of this approach is that instead of assessing a particular strategy on the

0-8186-2310-1/91 $1.00 © 1991 IEEE

basis of a selected set of experiments, it can be analyzed in a variety of different conditions.

2. Related Work

Scheduling strategies can be either static or dynamic. In the former case, work load allocation decisions are taken before run time and tasks are assigned to individual processors of the system. Static scheduling strategies can generate good load balance and are useful for workload with static structures [17]. However, if the application have dynamic structures or if the tasks are created at run time [9], dynamic scheduling is required [5], [8]. It is essential for a dynamic scheduling strategy to balance computation load by migrating workload from the heavily loaded processors to the lightly loaded or idle processors of the system. Dynamic task scheduling or load balancing, on a multicomputer network consisting of autonomous computers connected together via communication links, has been extensively investigated by many researchers [5], [7], [8], [11], [12]. Dynamic scheduling can be centralized in which the control is in the hand of a single node. In a decentralized approach [5], [6], [7], this control is distributed amongst all the nodes in the system. In contrast ,in a semi–distributed control, only a subset of nodes make scheduling decisions [1]. For decentralized control, various algorithms based on bidding [14], and nearest neighbors load balancing have been proposed [8] , [11], [13]. These strategies are also classified according to the type and amount of information exchange [8].

3. The Task Scheduling Algorithm

As discussed above, an efficient dynamic scheduling strategy is one that is stable, returns a fast response time, and incurs a low overhead. In general, a scheduling strategy consists of three policies, namely transfer policy, location policy and information policy [5]. The information exchange policy in our case is that when the scheduler of some node needs to schedule some task, it collects the load status of its neighbors.

Since information interchange and execution of scheduling algorithm takes certain amount of time, the tasks arriving during that time wait in a waiting queue. For each communication link, a communication queue is maintained. At each node, task transferred from other node joins the locally generated tasks, and both are handled with equal priority. If a task is decided to be scheduled locally, it is entered in the execution queue which is served by the CPU on the FCFS basis. The length of this queue represents the load of a node. A task may migrate from node to node in the network before finally being executed at some node. Figure 1 illustrates such a multicomputer network.

The proposed algorithm is a combination of neighborhood averaging and bidding approach. When a task is to be scheduled, the scheduler broadcasts messages to its neighbors asking for bids. A neighbor calculates the average load of its own neighborhood and if its own load is less than that average, it sends a *yes* message along with its load information to the requesting node. After the requesting node has received all the bids, it calculates the average load of its own neighborhood. If its local load is greater than average, it selects the node with minimum load out of those neighbor which sent yes messages. If the local execution queue is empty or the local load is less than average and none of the neighbors reply with yes messages, the task is scheduled in the local queue. This strategy is proposed to add more stability to neighborhood averaging strategy. This extra level of stability is due to the fact that the receiving node expresses its willingness to receive a task only if its load is less than its own neighborhood average. This reduces the number of extra task migrations and hence task thrashing. Avoidance of unnecessary task migration is particularly important at high load when every overloaded node tries to get rid off its load by assuming that other nodes are lightly loaded. As shown later, at high load, the task migration probability of the proposed algorithm is greatly reduced but at the same time it is large enough to transfer extra load. This algorithm is termed as Bidd–Average.

For comparison, we select random scheduling strategy which also makes use of nearest neighbor load balancing. In this strategy, the task scheduler calculates the average of its own load and the load of all neighbors. If the local load is greater than the average, the task is sent to a randomly selected neighbor. The objective behind selecting this strategy is to show the tendency of random strategy to cause extra migrations.

4. Performance Modeling

In this section, we describe a performance evaluation model for scheduling strategies described above. First, we show that the class of distributed load balancing strategies described above can be modeled by an open central server queuing model. A multicomputer network can be represented as a graph where each node represents an independent processor and each edge represents the communication link between two nodes. We consider a multicomputer network in which processing nodes are connected with each other through a symmetric topology, that is, each node is linked to the same number of nodes. The number of links per node is called the degree of the network. The processors of the system can be heterogeneous but in this paper we consider only the homogeneous case. Symmetry implies that the interconnection network of the system is a regular graph with fixed number of links per node whereas homogeneity implies that the processors of the system are identical.

We assume that task arrive at each node with an average arrival rate of λ tasks per time–unit per node. The task arrival process is assumed to be Poisson and inter–arrival time is assumed to be exponentially distributed. The average scheduling, communication and execution times are denoted by $1/\mu_S$, $1/\mu_C$, $1/\mu_E$ time–units, respectively and all of them are assumed to be exponentially distributed. With

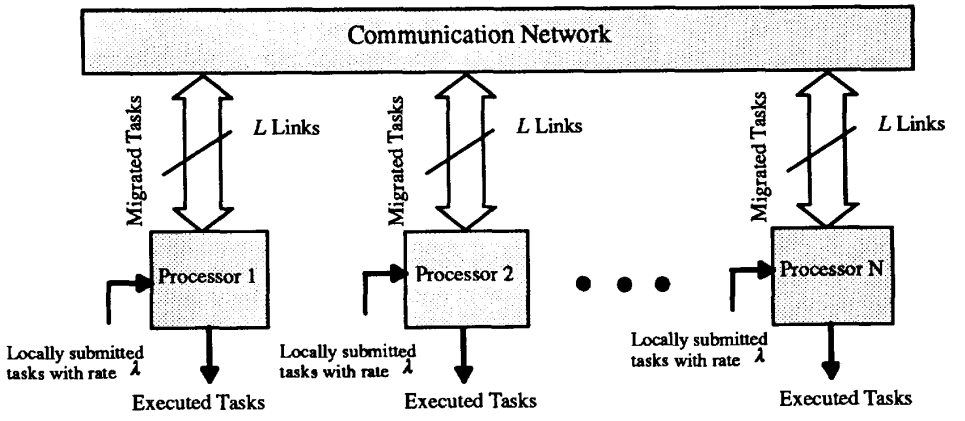

Figure 1: A logical view of decentralized task scheduling.

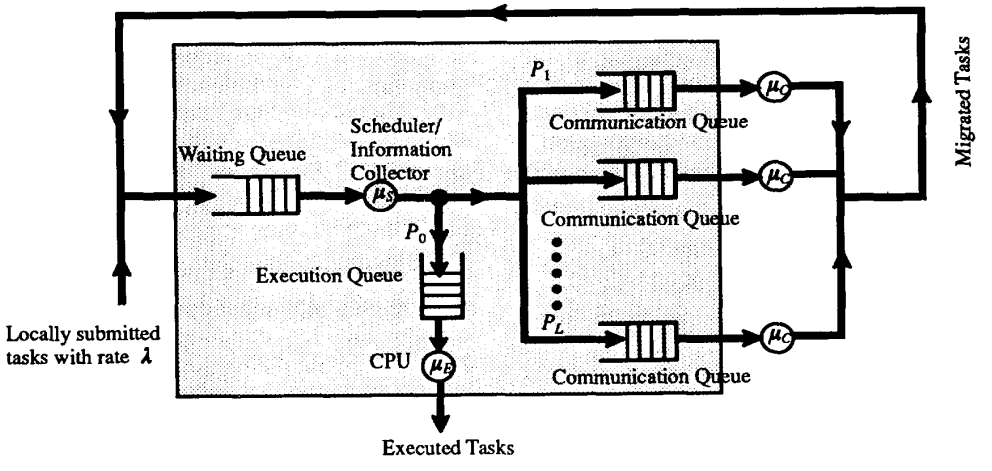

Figure 2: The system in Figure 1 represented by open network central server model.

nearest neighbor load balancing, the steady state departure and task arrival rates at a node are the same. When a task migrates from one node to another, it encounters a statistically identical node. Therefore, under steady–state conditions, the multicomputer network shown earlier in Figure 1 can be equivalently represented by the open central server queuing model as shown in Figure 2. The model consists of a waiting queue, L communication queue sand an execution queue. A task arriving at that node enters the local execution queue with probability, P_0 or migrates to one of the neighbors with probability $1 - P_0$. The model is approximate since routing of tasks is dependent on the state of execution queues. This model is, however, also validated through simulation results, which are obtained on actual network topologies.

We can view a task's residence time in the system as consisting of two phases. In the first phase, the task may keep on migrating during the course of which it waits in the waiting queue, gets service from the scheduler, waits in the communication queue, and then it may transfers to another node. If it is re–migrated, the same cycle can start all over again. The second phase starts when the task is finally scheduled at the execution queue of a node. The second phase includes the queuing and service time at the CPU. By solving this model [16], response times in the first phase can be obtained as:

Once a task is scheduled at a local execution queue, the duration of its residence time in the system, starting from the time it is scheduled to the time it finishes execution, can be calculated as:

$$(Resp.\ Time)_{phase1} = \frac{1/(P_0\mu_E)}{1-\lambda/(P_0\mu_E)} + \sum_{j=1}^{L} \frac{p_j/(P_0\mu_j)}{1-\lambda p_j/(\mu_j P_0)}.$$

$$(Resp.\ Time)_{phase2} = \frac{E[N_E]}{\lambda}.$$

where $E[N_E]$ is the average execution queue length. The complete response time, therefore, is given by:

$$Resp.\ Time = (Resp.\ Time)_{phase1} + (Resp.\ Time)_{phase2}.$$

The above equation implies that, for a given system load, μ_0 and μ_j 's, the response time yielded by a load balancing strategy can be calculated if the probability, P_0, and the average execution queue length, $E[N_E]$ is determined. The probability that a task will be migrated to another node is simply $1 - P_0$. The migration probabilities to individual channels at each node are identical. The probability, P_0, is calculated, from the simulation data, after dividing the average number of locally scheduled tasks by the total number of tasks arrived, at each node. The average execution queue length, $E[N_E]$, determines how smoothly load is balanced. Both parameters, P_0 and $E[N_E]$, depend on a number of system parameters such as λ, μ_S, μ_C, μ_E, and L. In the next sections, we briefly describe the simulation methodology which was used to obtain a very large data set from different test cases. We describe how we performed statistical analysis on the simulation data and determined the sensitivity of P_0 and $E[N_E]$ against different system parameters. Both task scheduling strategies were simulated. A long series of simulation runs was conducted to obtain a large number of data points for P_0 and $E[N_E]$ for each particular strategy by varying λ, μ_S, μ_C and number of links per node.

It is worth mentioning that the simulator takes into account the time to schedule a task which includes the exchange of state information and the execution of scheduling algorithm itself. Most previous studies have ignored this overhead. We have assumed an average scheduling time, $1/\mu_S$, which in turn can be normalized with respect to the execution time, μ_E. For example, if μ_S is 10 tasks/time–unit and μ_E is 1 task/time–unit, it means that the average task scheduling time is 1/10 of the execution time. We consider it an input parameter which can be observed from a real system depending upon how the information message handling and regular task migration is implemented.

We characterized P_0 and $E[N_E]$ in terms of system parameters such as λ, μ_S, μ_C and system network topology, with statistical analyses. For this purpose, a regression analysis was then performed to obtain a model that expresses P_0 in terms of the aforementioned parameters.

As explained earlier, the response time of a task consists of two parts. The first part is the response time before the task is scheduled in an execution queue. This is simply equal to the time at which the task is scheduled (in the execution queue of some node) minus its arrival time. This response time, called transient time, is completely described by P_0 which indicates the task migration tendency of a load balancing strategy. The second part of response time shows how much time (queuing delay plus execution time) a task takes to finish its execution, after eventually being scheduled. This time is equal to the time the task finishes execution minus the time at which it was scheduled in the execution queue. The best transient response time results when a strategy's P_0 is neither very high nor very low. In other words, the strategy should not have task thrashing tendency and yet it should make task migrations whenever appropriate. The second part of the response time depends on a strategy's load equalization ability, that is, it results in a smaller average execution queue length if the load is equally balanced. Both of these factors, however, are dependent on each other. For example, if a strategy suffers from task thrashing, execution queue length is not balanced and the average value of queue length increases.

$$P_0(Random) = \left[1 + e^{-\left(1.722 - 0.154 * links + 0.001 * \mu_C + 0.001 * \mu_S - 1.320 * \lambda\right)}\right]^{-1} \quad (1)$$

$$P_0(Bidd-Average) = \left[1 + e^{-\left(1.349 - 0.10827 * links + 0.006 * \mu_C + 0.013 * \mu_S - 0.794 * \lambda\right)}\right]^{-1} \quad (2)$$

$$E[N_E](Random) = \exp\left(-1.960 - 0.062 * links + + 3.300 * \lambda\right) \quad (3)$$

$$E[N_E](Bidd-Average) = \exp\left(-1.834 - 0.055 * links + + 2.913 * \lambda\right) \quad (4)$$

5. Results

First, we examine the task migration phenomenon associated with both strategies. This phenomenon is indicated by the P_0 which is the probability that a task is scheduled locally in the execution queue. The probability that a task is migrated to a neighbor is simply $1 - P_0$, as mentioned earlier. The probability P_0 becomes low if a scheduling strategy has a tendency to make more task migrations. In contrast, if a scheduling strategy tries to keep more tasks locally, P_0 becomes very high. For instance, P_0 will be equal to 1 if no load balancing is done at all.

The impact of system load on P_0 is significant because the load of a node as well as those of its neighbors affect the behavior of the scheduling algorithm. The curves for P_0 with varying system load are shown in Figure 3. For this case the average value of μ_E is selected to be 1 task per time units whereas both μ_S and μ_C are selected to be 20 tasks per time unit. An ideal strategy keeps keeps the value of neither very high nor very low and adjusts to the system load. The average arrival rate per node is varied from 0.3 to 0.9. The topology used in this case is a 16–node hypercube with 4 links per node. These results are obtained through the analytical model by using equation 1 and 2. A threshold can also be used to control P_0 by adapting a static transfer policy. Since both Random and Bidd–Average strategies determine their threshold dynamically depending upon their own load and their neighbors load, they adjust to the system load. However, random strategy can still make unnecessary task migrations which makes a task go through scheduling and communication delays. A neighboring node which appears relatively less loaded as compared to the local node may still be loaded enough to make the task wait. The Bidd–Average algorithm makes a double check to see if the neighboring node is worth receiving the task. Due to this reason, the value of P_0 is higher for the Bidd–Average algorithm (showing controlled task migrations due to lower value of $1 - P_0$).

The response time in the first phase should be reduced to minimum so that less time is spend in communication and scheduling but in order to reduce execution queue lengths, load balancing needs to be done by task migration. Once a task is scheduled at some node, the duration of the second phase of its response time is dependent on the execution queue length of that node. The average queue length versus system load is shown in Figure 4. The parameters selected for this figure are the same as those used for Figure 3. These results are computed using equations 3 and 4. Again, the proposed algorithm is shown to results in a smaller average queue length, despite its controlled task migration.

Next, we show the average response time in phase 1 and phase 2. These results are depicted in Figure 5 and Figure 6. These results are obtained through the analytical model. From these figures, we draw the following insights. The difference in response times of both strategies in phase 1 and phase 2 is not very high at low load. However, this difference becomes significant as the load increases. This indicates that Bidd–Average performs much better than the random strategy, as the load increases. The accumulative average response time, which is the sum of response times in both phases is shown in Figure 7. For this case, we present the results from the analytical model as well as simulation. Simulation results are obtained by running these algorithms on actual topology (16–node hypercube). We note that the random algorithm performs better at low loading conditions but its performance deteriorates with an increase in load. We also observe that the performance model alone is able to produce distinct and accurate results, and the difference in the performance of both algorithms is clearly detected by the model. Moreover, the model is able to produce non–linear curve for response time versus system load which is equally matched by simulation. These results validate our performance model.

As opposed to some of the previous studies, we explicitly take into account the cost of task scheduling overhead and the cost of task migration. Moreover, since both scheduling and communication processes take certain amount of time, waiting queues are maintained for tasks arriving during those times. These queues are taken into account in the analytical model, as shown earlier in Figure 2, and are also implemented in the simulation. In order to evaluate the performance of the proposed algorithm and check the validity of the proposed model for various parameters, we change μ_S and μ_C but keep the rest fixed. The average accumulative time of a task is plotted in Figure 8 against variable task transfer rate. The load per node is set to be 0.8. The scheduling rate is fixed as 20 tasks per unit–time and the task transfer rate is varied from 4 to 40 tasks per unit–time, representing slow to very fast communication conditions. Simulation results are also presented and compared with the results of the analytical model. Under these conditions, the proposed algorithm is again shown to outperform Random algorithm. Moreover, the model is, again, shown to predict the average response time which closely matches the response time produced by simulation.

The task scheduling time has greater impact on the average task response time than the task communication time. This is obvious when we compare the results given in Figure 8 to those in Figure 9, indicating that the average response time with slow scheduling rate and high communication rate (Figure 9) is greater than the response time with fast scheduling rate and slow communication rate (Figure 8). The observation is true for both strategies. However, if both μ_C and μ_S are low, a significantly higher response time will result.

Next, we show the impact of system topology on the average response time. In addition to 32–node hypercube with 5 links per node, we consider a 8–node folded hypercube.

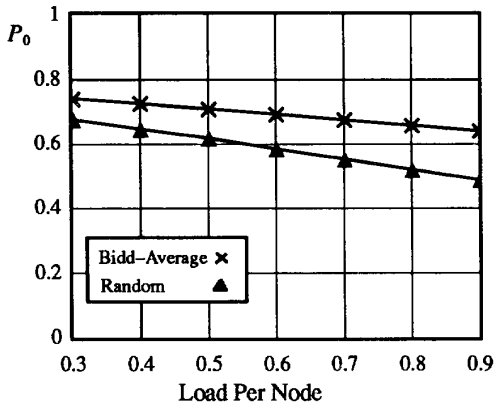

Figure 3: Probability that a task is scheduled locally.

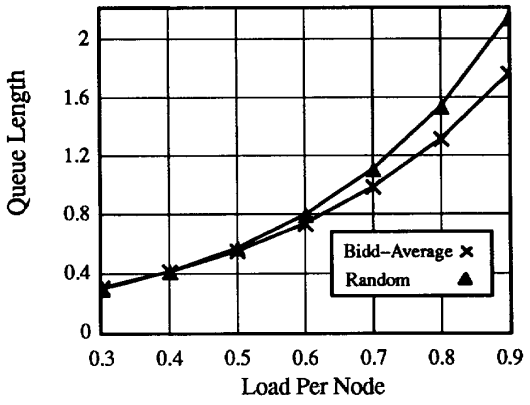

Figure 4: Average Execution Queue length.

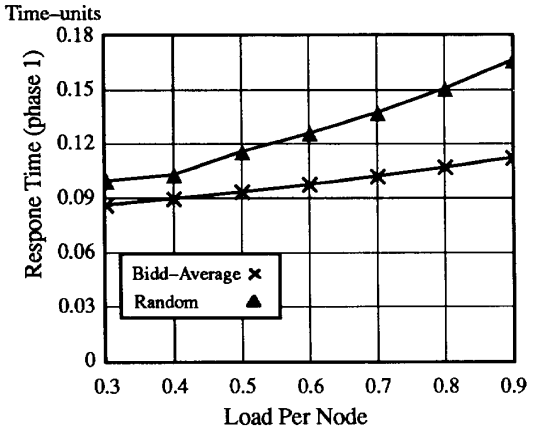

Figure 5: Average task settling time.

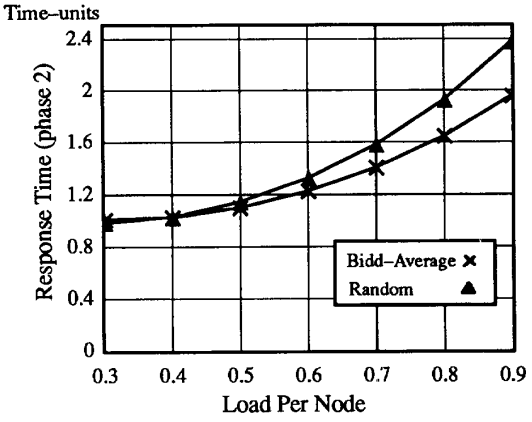

Figure 6: Average task waiting time plus execution time .

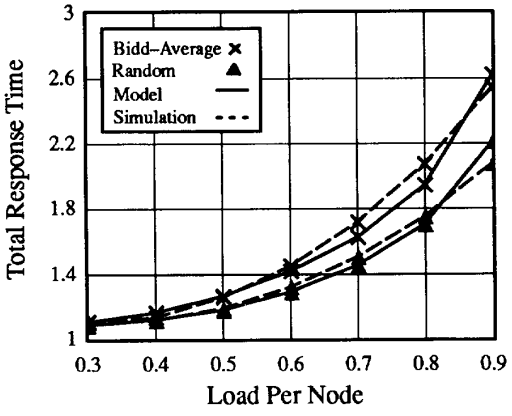

Figure 7: Average accumulative task response time versus system load.

This topology is similar to 8–node hypercube except that it has an extra link at every node which connects it to the node whose binary address is the complement of this node. Therefore, a 8–node folded hypercube has 4 links per node, as opposed to 3 links per node for 8–node hypercube. The second selected topology is a 9–node mesh with 4 links per node, with wrapped around edges. The other two topologies are a 8–node fully connected network with 7 links per node and 16–node chordal ring with 3 links per node. The communication rate and scheduling rate are both 20 tasks/time–unit and load per node is 0.7. The results for these topologies are given in Figure 10, showing the average accumulative response time obtained with both analytical model and simulation. The percentage difference in results of analytical model and simulation is also indicated. We make the following observations from this figure. The proposed algorithm performs better on all of these topologies. The difference in the response time of the model and simulation is again very small. For both strategies, the fully connected network performs the best whereas the ring topology per-

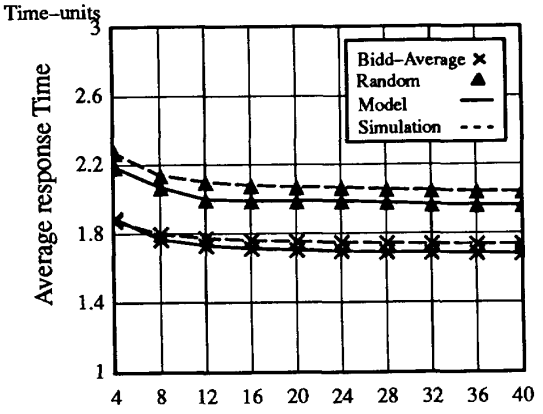

Figure 8: Average accumulative task response time versus task transfer rate.

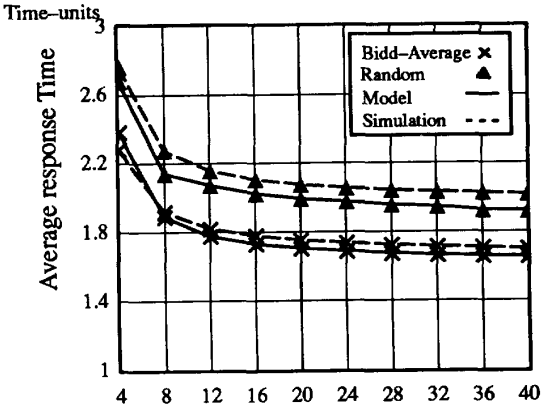

Figure 9: Average accumulative task response time versus task scheduling rate.

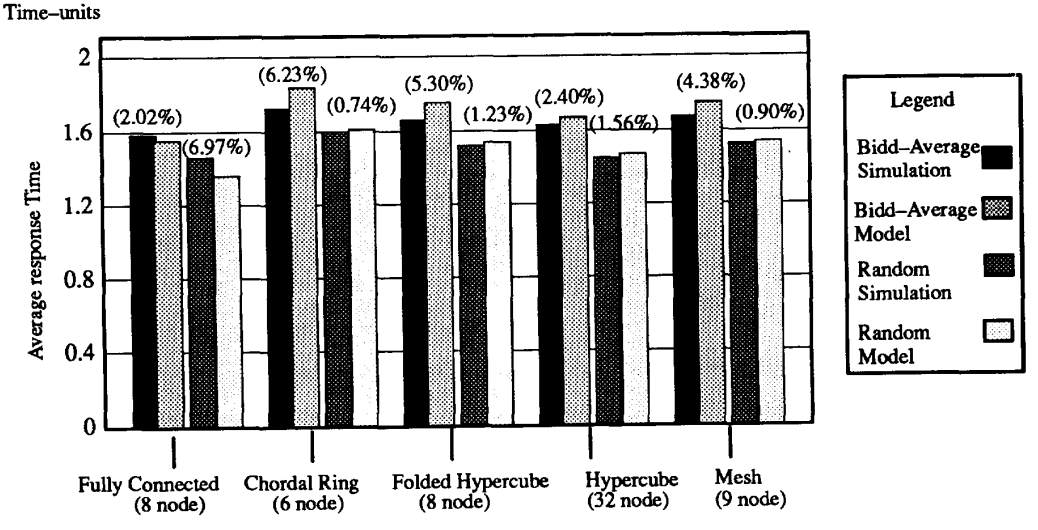

Figure 10: Comparison of response times predicted by the model and simulation for some more topologies of varying sizes (λ = 0.7 task/time–unit, μ_C =16 task/time–unit and μ_S = 16 task/time–unit).

forms the worst which implies that greater number of links per node result in better load balancing. This is also intuitively true since the availability of more links implies an increased probability of finding suitable neighbors for task migration.

The is no difference in the performance of folded hypercube and mesh – both of these topologies have 4 links per node for 16–node network. The number of links per node determines the impact of topology on a scheduling strategy. It is because of this reason that the performance of both topologies is the same as 9–node mesh (4 links per node). In other words, the scheduler at the nodes of both topologies perform logically the same and the topological effect on nearest neighbor load balancing can indeed be modeled by the number of links per node.

From this figure, the validity of the performance model is more strongly established as we obtain response time from the model and compare it with some additional simulation runs. For comparing both results, the empirical data obtained from these simulation runs has not been used for statistical modeling. The difference between any pair of data sets does not exceed $\pm$ 7 %..

6. Summary

In this paper, we have proposed a task scheduling algorithm and have evaluated its performance via analytical modeling and simulation. The proposed algorithm performs load balancing among nearest neighbors and has been shown to yield a good performance. Using the performance evaluation approach, we are able to compare two different load balancing schemes on a unified basis. We have shown that these algorithms can be modeled by an open central server queuing network if the system is symmetric and homogeneous. The statistical characteristics of the proposed algorithm are presented by showing the sensitivity of its queuing parameters with respect to various system parameters. By considering examples from a wide range of system parameters, it is shown that the average task response time computed by the performance model closely matches the response time obtained via simulation. For dynamic scheduling strategies, the response time of a task can be analyzed in two phases. By comparing the algorithm with random strategy, we notice that the proposed algorithm exhibits extra stability and avoids unnecessary task migrations.

References

[1] Ishfaq Ahmad and Arif Ghafoor, "A Semi Distributed Task Allocation Strategy for Large Hypercube Supercomputers," in *Proc. of Supercomputing '90*, Nov. 1990, pp. 898–897.

[2] Ishfaq Ahmad, Arif Ghafoor and Kishan Mehrotra, "Performance Prediction for Distributed Load Balancing on Multicomputers," Technical report + SU–CIS–91–12, Department of Computer Science, Syracuse university, Syracuse, New York.

[3] Rafael Alonso and Luis L. Cova, "Sharing Jobs Among Independently Owned Processors," in *Proc. of 8–th Intl. Conf. on Distributed Computing Systems*, 1988, pp. 282–288.

[4] Yuan–Cheih Chow and Walter H. Kohler, "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *IEEE Trans. on Computers* ,vol. c–28, no. 5, May 1979, pp.354–361.

[5] Derek L. Eager, Edward D. Lazowska and John Zahorjan,"Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. on Software Eng.* ,vol. SE–12, pp. 662–675, May 1986.

[6] Kemal Efe and Bojan Groselj, "Minimizing Control Overhead in Adaptive Load Sharing," in *Proc. of 9–th Intl. Conf. on Distributed Computing Systems*, 1989, pp. 307–315.

[7] Arif Ghafoor and Ishfaq Ahmad "An Efficient Model of Dynamic Task Scheduling for Distributed Systems, "in Proc. of *COMPSAC '90*, Oct., 1990, pp.442–447.

[8] Dirk C. Grundwald, Bobby A. A. Nazief and Daniel A. Reed, "Empirical Comparison of Heuristic Load Distribution in Point–to–Point Multicomputer Networks," *Proc. of The Fifth Distributed Memory Computing Conference*, April 1990, pp. 984–993.

[9] Jeff Koller, "The MOOS II Operating System and Dynamic Load balancing," in *Proc. of The Fourth Conf. on Hypercube, Concurrent Computers and Applications*, March 1989, pp. 599–602.

[10] Andrew Ross and Bruce McMillin, "Experimental Comparison of Bidding and Drafting Load Sharing Protocols," *Proc. of The Fifth Distributed Memory Computing Conference*, April 1990, pp. 968–974.

[11] Vikram A. Saltore, "A Distributed and Adaptive Dynamic Load Balancing Scheme for Parallel Processing of Medium–Grain Tasks," *Proc. of The Fifth Distributed Memory Computing Conference*, April 1990, pp. 994–999.

[12] Niranjan Sivrati and Philip Krueger, "Two Adaptive Location policies for Global Scheduling Algorithms," *Proc. of The 10–th Int' l conf. on Distributed Computing systems* , June 1990, pp. 502–509.

[13] Wei Shu and L. V. Kalé, "A Dynamic Scheduling Strategy for the Chare–Kernel System," in *Proc. of Supercomputing '89*, November 1989, pp. 389–398.

[14] John A. Stankovic and I. S. Sidhu, "An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups, "in *Proc. of 4–th Int' l. Conf. on Distributed Computing Systems*, 1984, pp. 49–59.

[15] Anders Svensson,"Hostory, an Intelligent Load Sharing Filter," in *Proc. of 10–th Intl. Conf. on Distributed Computing Systems*, 1990, pp. 546–553.

[16] Kishor S. Trivedi, *Probability & Statistics with Reliability, Queuing and Computer Science Applications*, Prentice–Hall, inc. Englewood Cliffs, NJ, 1982.

[17] Min–You Wu and W. Shu, "Scatter Scheduling for Problems with Unpredictable Structures," in Proc. of The Sixtth Distributed Memory Computing Conference. April 1991, pp. 137–143.