Performance Prediction of Distributed Load Balancing on Multicomputer Systems

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

This paper presents a performance evaluation approach to compare different distributed load balancing schemes on a unified basis. This approach is an integration of simulation, statistical and analytical models, and takes into account the fundamental system parameters that can possibly affect the performance. We show that all the sender–initiated distributed load balancing strategies can be modeled by a central server open queuing network. Furthermore, these load balancing strategies can be characterized by only two queuing parameters – the average execution queue length and the probability that a newly arrived task is executed locally or migrated to another node. To capture the relation between these queuing parameters and various system parameters, a statistical analysis has been carried out on the empirical data obtained through simulation. The analytical queuing model is then used to predict the response time of a system with any combination of system parameters. Experimental results are obtained for six different load balancing strategies. The proposed model provides performance results in a straightforward manner and can be beneficial to the system designers in assessing the system under varying conditions.

1. Introduction

Efficient utilization of a multicomputer system lies in its ability to efficiently partition and balance the computational load among its computing nodes. With the increasing popularity of multicomputer systems, researchers and system designers have been focusing on these essential issues. If the decisions to allocate workload tasks to processing nodes are fixed and are taken before actually running the problem, then load balancing is considered static. For dynamic load balancing, there are no fixed allocation decisions and load is balanced depending upon the time dependent state of the system. Dynamic load balancing has also been termed as dynamic load sharing [3], or load distribution [8]. As noted in [3], any simple dynamic load balancing algorithm improves the performance of the system, and is better than no load balancing. Dynamic load balancing strategies are characterized by the manner in which information exchange and control of workload allocation takes place. The control can be centralized [14], fully distributed [2], [3], [4], [9], [12], [15], [22] or semi distributed [1]. With fully distributed control, the load balancing strategy is incorporated at every node of the system in that each node in the system makes autonomous decisions. A node decides whether the task submitted to it should be executed locally or transported to some other node. If the task should be migrated, the local node needs to know the load status of other nodes. A node for task migration can be selected randomly [3], [6], [22] or with some other criteria [3]. However, the accuracy of scheduling decisions in decentralized algorithms, depends on the accuracy and amount of state information [8].

Wang and Morris [23] proposed a number of relatively simple load balancing algorithms and classified them into two categories: source–initiated and server–initiated. In a source–initiated algorithm, tasks enter the distributed system via source nodes and are processed by server nodes. Fox *et al.* [5] presented a load balancing scheme by making use of the analogy of load balancing to minimizing an appropriate energy function. In [15] and [20], various bidding algorithms have been proposed, which belong to the sender–initiated class. A drafting algorithm belongs to the server–initiated class [13]. A comparison of these two types of algorithms [16] reveals that in spite of the fact that the bidding algorithm suffers from *task–dumping* or *task–thrashing*, it performs consistently better than the drafting algorithm. Task–thrashing is a phenomenon associated with load balancing where a lightly loaded node can become a victim of task arrivals from other nodes [6], [12]. Load balancing algorithms can also suffer from *state woggling* – another performance decaying phenomenon in which processors frequently change their status between low and high [16].

For systems with certain interconnection topologies, distributed load balancing schemes based on task migration among nearest neighbors have gained considerable attention. In a number of independent studies [6], [8], [10], [17], variants of this strategy have been proposed and their effectiveness has been proven both by simulation and implementation observations. Kalé [17] has compared one version of this strategy, known as Contracting Within Neighborhood (CWN), to the Gradient Model [11] and has shown that CWN spreads the load more quickly and performs better. In two more studies [6], [17], the concept of load averaging among neighbors is introduced. The advantage of load averaging is that each node tries to keep its own load equal to the average load among its nearest neighbors. Shu and Kalé [19] have proposed and implemented a revised version of CWN known as Adaptive Contracting Within Neighborhood (ACWN), which consistently shows better response time compared to the Gradient model and Random strategy. Grunwald *et al.* [8] have proposed a classification scheme for the type of information required to make load balancing decisions.

Given the diversity of a number of proposed strategies and their dependence on a number of parameters, it is difficult to compare their effectiveness on a unified basis. One particular strategy may perform well under a certain combination of parameters, such as, system load or system communication rate on a certain topology. The same strategy may be outperformed by another strategy due to a difference in information collection and scheduling overhead. In addition, simplified assumptions and neglecting important parameters sometimes obscures the relative merits and demerits of each strategy. This paper presents an approach to predict and compare the performance of different load balancing schemes based on a unified basis. Our approach, which is an integration of simulation, statistics and analytical models, takes into account various system parameters, such as system load, task migration time, scheduling overhead and system topology etc., that can affect the performance. We show that load balancing strategies, belonging to the sender–initiated class, can be modeled by a central server queuing network. We also show that these strategies can be characterized by only two parameters – the average queue length and the probability that a task is executed locally or migrated to another node. Through an extensive simulation, a large number of values of the average queue length and the probability associated with task migration have been obtained. A statistical analysis has been performed on these data points to capture the relation between the queuing parameters and the system parameters. The analytical queuing model is then used to predict the response time of a system with any set of parameters. Six different load balancing algorithms have been studied and characterized.

This performance prediction approach has many advantages. First, instead of assessing a particular strategy on the basis of a selected set of experiments, any combination of parameters can be used to predict the performance. Second, all strategies can be relatively compared by selecting more appropriate and realistic parameters. Finally, an existing system can be tuned, and a system design can be evaluated before it is actually built. The response time predicted by the model is compared with the response time produced by simulation for all six strategies.

2. Selected Load Balancing Strategies

We consider a fully homogeneous multicomputer system 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, called the degree of the network, is considered as one of the system parameters and is denoted as *L*. The workload submitted to the system is assumed to be in the form of tasks, which are submitted to each node with an average arrival rate of λ tasks per time–unit per node. The task arrival process is assumed to be Poisson. The load balancing control is fully distributed for which each node makes an autonomous decision to schedule a task by collecting the load status information from its neighbors. A task is either scheduled to a local execution queue or it is migrated to one of the neighbors connected with each communication channel. The information and scheduling takes a certain amount of time, which is assumed to be exponentially distributed with an average of $1/\mu_S$ time–units. Information is collected by a hardware/software component at each node and is called Collector/Scheduler.

Since information interchange and execution of the scheduling algorithm takes a certain amount of time, the tasks arriving during that time wait in a waiting queue. For each communication link, a communication queue is maintained. The underlying network supports point–to–point communication and the communication channel is modeled by a server. A communication server transfers a task from one node to another with an average of $1/\mu_C$ time–units. The task communication time is also assumed to be exponentially distributed. Each communication queue is served on the FCFS basis. At each node, the incoming traffic from other nodes joins the locally generated traffic, and both are handled with equal priority. Each node maintains an execution queue in which locally scheduled tasks are served by a CPU on the FCFS basis . A task may migrate from node to node in the network before finally being executed at some node. The execution time is also assumed to be exponentially distributed with an average of $1/\mu_E$ time–units.

We have analyzed six different sender–initiated load balancing strategies for varying information collection mechanisms and scheduling disciplines. Based on the information interchange mechanism, these strategies can be further classified into two categories. In the first category, the information about the load and the status of other nodes is collected at the time a task is scheduled for execution or migration. The load is expressed in terms of the length of the execution queue. This load metric has been widely accepted and experimental results have shown that it accurately reflects the CPU load [16]. In the second category, nodes exchange the load information among their neighbors periodically. Within each category, we have considered three different scheduling policies.

Category I: Information Exchange at the Time of Task Schedule

- **FRandom:**

  In this strategy, the task scheduler calculates the average of the local 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. If the local execution queue is empty (or local load is less than the average), then the task is sent to the local execution queue.

- **FMin:**

  In this strategy, the task scheduler sends a new task to the node that has the minimum load. However, if the local node's load is equal to the minimum load among neighbors, the local node is given priority.

- **FAverage:**

  In this strategy, the task scheduler calculates the average of all neighbors' load and its own load. If the local load is greater than the average, the task is sent to the neighbor with the minimum load. However, if the local execution queue is empty or the local load is less than the average, the task is sent to the local execution queue.

Category II: Periodic Information Exchange

- **PRandom:**
  This strategy is similar to *FRandom* except that every node sends it own load information to all its neighbors periodically. The time period, T_u for sending messages is a system parameter.
- **PMin:**
  This strategy is similar to *FMin* except that information exchange is done periodically.
- **PAverage:**
  This strategy is similar to *FAverage* except that information exchange is done periodically.

3. The Performance Prediction Model

In this section, we describe the performance prediction model for the distributed load balancing strategies described above. This model is an integration of simulation, and statistical and queuing models. First, we describe the queuing model and show that the class of distributed load balancing strategies described above can be modeled by an open central server queuing model.

3.1. The Queuing Model

As described above, the multicomputer system considered here is symmetric and homogeneous. By symmetry, we mean that the interconnection network of the system is a regular graph with a fixed number of links per node. By homogeneity we imply that the processors of the system have identical processing speeds. Similarly, all communication channels and task schedulers are identical. The steady-state task departure and arrival rates at every node are the same. As explained earlier, a task keeps on migrating until it finds a suitable node. When a task migrates from one node to another, it sees a statistically identical node. Therefore, the steady-state behavior of nearest neighbor load balancing can be approximated by the open central server queuing model as shown in Figure 1. The model consists of a waiting queue, L communication queues and an execution queue. This model is approximate, since routing of tasks is dependent on the state of execution queues. However, as described in the next section, simulation results obtained on actual network topologies are very close to the analytical results determined from this model, which validate that the proposed model of Figure 1 indeed represents the task scheduling and migration process.

The duration of a task's residence time in the system consists 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 transfers to another node. At that point the same cycle may start all over again. Once the task is scheduled at the execution queue of a node, the second phase starts, which includes the queuing and service time at the CPU. In the first phase, the task can be viewed as occupying either the task scheduler or one of the communication links. The Markov chain shown in Figure 2 describes the behavior of the central server which in turn explains the task migration phenomenon before the task enters the execution queue. The state of the Markov chain is described by $(L + 1)$ tuple, $k_0, \ k_1, \ . \ . \ k_L$ in which k_i represents the number of tasks at the i-th queue ($0 \ \leq \ i \ \leq \ L$) at a node.

It follows [21] that the model can be solved by the Jacksonian network, which has the product form solution; that is, the joint probability of k_j tasks at queue j ($j = 0, 1, ..., L$) is given by the product :

$$p(k_0, k_1, \ , k_2, \ . \ . \ . k_L) \ = \ \prod_{j=0}^{L} p_j(k_j)$$

where $p_j(k_j)$ is the probability of k_j tasks at j-th queue and is given by:

$$p_j(k_j) \ = \ (1 - \varrho_j) \ \varrho_j^{k_j}.$$

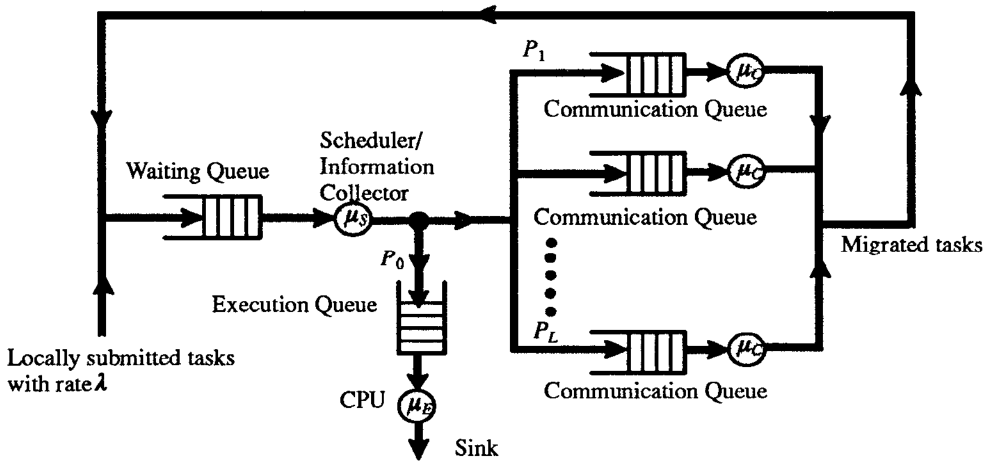
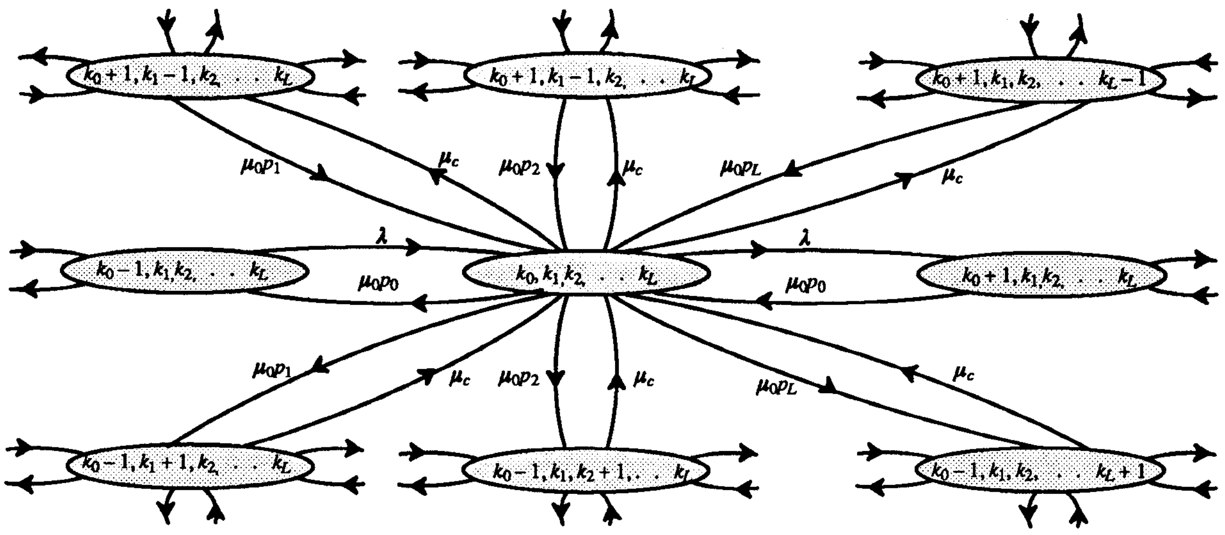


Figure 1: Distributed load balancing represented by open central server model.



$k_0+1, k_1-1, k_2, \ldots k_L$
$k_0+1, k_1-1, k_2, \ldots k_L$
$k_0+1, k_1, k_2, \ldots k_L-1$
$\mu_0 p_1$
μ_c
$\mu_0 p_2$
μ_c
$\mu_0 p_L$
μ_c
λ
λ
$k_0-1, k_1, k_2, \ldots k_L$
$\mu_0 p_0$
$k_0, k_1, k_2, \ldots k_L$
$\mu_0 p_0$
$k_0+1, k_1, k_2, \ldots k_L$
$\mu_0 p_1$
μ_c
$\mu_0 p_2$
μ_c
$\mu_0 p_L$
μ_c
$k_0-1, k_1+1, k_2, \ldots k_L$
$k_0-1, k_1, k_2+1, \ldots k_L$
$k_0-1, k_1, k_2, \ldots k_L+1$

Figure 2: Markov chain with the state of the chain describing the number of tasks at each queue of a node.

For the j-th component, the average utilization, ϱ_j, is equal to λ_j/μ_j. The equation implies that the lengths of all queues are mutually independent in a steady state. The above model can also be solved while considering the probabilistic behavior of a task. Suppose, after the task is served by the scheduler, it goes to the i-th link with probability P_i or it enters the local execution queue with the probability P_0. When a task leaves (enters) the waiting queue, the number of tasks in that queue is decreased (increased) by one. Similarly, when a task is served by the communication, a statistically identical task joins the waiting queue. The average queue length and average response time for the j-th component is given by:

$$E[N_j] = \frac{\varrho_j}{1-\varrho_j} \quad \text{and} \quad E[R_j] = \frac{1}{\lambda}\frac{\varrho_j}{1-\varrho_j},$$

respectively. The average number of tasks at a node is the sum of the average number of tasks at each component of a node and is given by:

$$E[N] = \sum_{j=0}^{L} E[N_j] = \sum_{j=0}^{L} \frac{\varrho_j}{1-\varrho_j},$$

from which the average response time before the task is scheduled in the execution queue can be computed as [21]:

$$E[R_S] = \frac{1}{\lambda}\sum_{j=0}^{L}\frac{\varrho_j}{1-\varrho_j}$$
$$= \frac{1\ /\ (P_0\mu_0)}{1\ -\ \lambda\ /\ (P_0\mu_0)} + \sum_{j=1}^{L}\frac{p_j\ /\ (P_0\mu_j)}{1\ -\ \lambda p_j\ /\ (\mu_j P_0)}$$

Once a task is scheduled at a local execution queue, the response time from the time it is scheduled to the time it finishes execution is given by:

$$E[R_E] = \frac{E[N_E]}{\lambda},$$

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

$$E[R] = E[R_S] + E[R_E],$$

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. In other words, P_0, is the probability with which a load balancing strategy schedules the tasks locally. The probability that a task will be migrated to another node is simply $1 - P_0$ and migration probabilities to individual channels at each node are identical. The average execution queue length, $E[N_E]$, determines how smoothly the load is balanced. Both parameters, P_0 and $E[N_E]$, depend on system parameters, such as λ, μ_S, μ_C, μ_E, T_u and L. In the next sections, we briefly describe the simulation methodology that is 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.

3.2. The Simulation Model

The above mentioned load balancing strategies were simulated on an Encore Multimax. The simulator accepts the topology of the network along with λ, μ_S, μ_C, μ_E, length of simulation run, and choice of load balancing strategies and their associated parameters. The results produced by the simulator include average response time, utilization of individual nodes, average time spent in communication, average number of messages, throughput, average number of migrations made by a task and their distribution, average lengths of waiting, communication and execution queues. In addition to average values, the variance and each node's individual statistics are also produced. The probability, P_0, is then calculated by dividing the average number of locally scheduled tasks by the total number of tasks arrived, at each node. The important aspects of discrete–event simulation are that it should be run for a sufficiently long time and initial transients should be removed before starting the accumulation of statistics. Moreover, the confidence interval must be calculated after running the same experiment with multiple independent streams. All of these features have been incorporated in the simulator and all results are obtained with a 99 % confidence interval.

A long series of simulation runs was conducted to obtain a total of 500 data values for P_0 and $E[N_E]$ were obtained, for each strategy. Three different topologies were selected, which included the ring, the hypercube and the folded hypercube [7], each consisting of 16 nodes. Each point for one particular strategy was obtained on each of the topologies by fixing one parameter and varying the rest. In most cases, λ was varied from 0.3 to 0.9 tasks per time–unit, μ_S was varied from 8 to 16 tasks per time–unit and μ_C was varied

Table I: Estimation for P_0 and its sensitivity versus system parameters.

Strategy	R–Square	System Parameter	Coefficient Estimate
FRandoom	0.9277	α	1.72220
		Links	–0.15421
		μ_C	0.00116 +
		μ_S	0.00140 ∓
		λ	–1.32043
FMin	0.9505	α	3.39618
		Links	–0.02139
		μ_C	0.00881
		μ_S	0.00841
		λ	–3.02439
FAverage	0.8668	α	1.48038
		Links	–0.09421
		μ_C	0.00839
		μ_S	0.01214
		λ	–1.04726
PPandom	0.9356	α	1.63440
		T_U	–0.13364
		Links	–0.13337
		μ_C	–0.00214+
		μ_S	–0.00395
		λ	–1.23230
PMin	0.9683	α	4.06852
		T_U	–0.21302
		Links	–0.10590
		μ_C	0.00013 +
		μ_S	0.00300
		λ	–3.48994
PAverage	0.9038	α	2.16996
		T_U	–0.42800
		Links	–0.14715
		μ_C	–0.00146+
		μ_S	–0.00356
		λ	–1.46303

Note: All estimates of model parameters are statistically significant, except slightly significant∓), and not significant (+).

from 8 to 16 task per time–unit. The task execution rate, μ_E, was fixed as 1 task per time–unit in all cases. For strategies that required a periodic information update, the update time, T_u , period was varied from 0.5 time–units to 1.5 time–units.

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 the 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. In other words, when μ_S is 10 tasks/time–unit and μ_E is 1 task/time–unit, 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.

3.3. Statistical Analysis

To characterize P_0 and $E[N_E]$ in terms of system parameters, such as λ, μ_S, μ_C, T_u and system network topology, statistical analyses have been performed. As described above, data on P_0 was collected for various values of the system parameters for each load balancing strategy. A regression analysis was then performed to obtain a model that expresses P_0 in terms of the system parameters. It is observed that the model shown in Equation 1 works quite well for all six strategies. The estimates of α_p and coefficients, β 's, are given in Table I along with measures that describe how well the above model predicts the observed P_0. For instance, in case of FRandom, the R–Square value is 0.9277, which implies that the regression model is able to compute 92.77 % of variation observed values of P_0. A similar regression analysis approach is taken to characterize $E[N_E]$ in terms of system parameters. In this case, the observed model is given by Equation 2. This model fits extremely well, as is observed from its R–Square values (all R–Square values are 99 %) given in Table II. In case of $E[N_E]$, the coefficients for μ_S and μ_C were found insignificant and hence are ignored.

3.4. The Complete Model

The complete model for performance prediction is shown in Figure 3. The performance measure is the average task response time. As described above, the model building consisted of running a large number of simulations, then applying statistical analysis to obtain models for P_0 and $E[N_E]$. Using this model, the values of P_0 and $E[N_E]$ can be directly computed for any of the six load balancing strategies with any combination of system load, communication rate, task scheduling rate, load update period (for load balancing strategies belonging to category II) and network topology. We then compute the average response time by using the formula given in section 3.1.

As explained earlier, this response time consists of two parts. The first part is the average response time before a

$$P_0 = \left[1 + e^{-\left(\alpha_p + \beta_{1p} links + \beta_{2p}\mu_C + \beta_{3p}\mu_S + \beta_{4p}\lambda + \beta_{5q}T_u\right)}\right]^{-1}$$ Equation (I)

$$E[N_E] = \exp\left(\alpha_q + \beta_{1q} links + \beta_{2q}\mu_C + \beta_{3q}\mu_S + \beta_{4q}\lambda + \beta_{5q}T_u\right)$$ Equation (II)

Table II: Estimation for $E[N_E]$ and its sensitivity versus system parameters.

Strategy	R–Square	Parameter	Parameter Estimate
FRandoom	0.9931	α	–1.96055
		Links	–0.06211
		λ	3.30085
FMin	0.9926	α	–1.67735
		Links	–0.03932
		λ	–2.86089
FAverage	0.9945	α	–1.92121
		Links	–0.05420
		λ	3.07851
PRandom	0.9981	α	–1.86021
		T_U	–0.02244
		Links	–0.05787
		λ	3.08250
PMin	0.9953	α	–1.59469
		T_U	0.01106
		Links	–0.03587
		λ	2.70488
PAverage	0.9984	α	–1.85252
		T_U	–0.00950
		Links	–0.05200
		λ	2.94949

task is scheduled in an execution queue. This is simply equal to the time the task is scheduled (in the execution queue of a node) minus the task 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 average response time shows how much time (queuing delay plus execution time) a task takes after eventually being scheduled. This time is equal to the time the task finishes execution minus the time the task 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 a 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, a smaller average execution queue length will result if the load is equally balanced. Both 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.

As an example, Figure 4 shows the plot of P_0 versus system load for all six strategies, on a 16–node hypercube. We notice that at low load both FMin and PMin have high values of P_0, which sharply increase at high load. This implies that both Min strategies schedule more tasks locally (and hence, make less migrations) but transfer more tasks at high load. In contrast, both 'random' strategies have low values of P_0, which implies that greater task migration takes place using random algorithms. Figure 5 shows the variations in $E[N_E]$ versus system load for all six strategies. From this figure, we observe that, in this case, the value of $E[N_E]$ is the minimum with *FAverage* followed by *PAverage*, and *PMin* results in the largest average queue length.

4. Performance Prediction, Evaluation and Comparison

After obtaining response time data from the performance prediction model, we compare it with the observed simulation results. Six load balancing strategies along with varying values of λ, μ_S, μ_C, T_u and different network topologies provide a wide range of figures to make a comparison between the response time obtained with the model and the response time obtained with simulation. However, we compare the two figures by varying one parameter while keeping the rest constant. The results are quite encouraging, and the difference between the two figures is found to be less than $\pm$ 7%. Since all results cannot be provided within this paper, we present only those results with a noticeable impact of each parameter on response time produced by the model, as well as by the simulation.

First, we examine the impact of system load on the average response time for all six strategies, shown in Figure 6 and Figure 7. In both figures, we have plotted the pairs of average response time computed from the model and the average response time observed from simulation. The difference in model and simulation results is also indicated on these figures. The task scheduling rate, μ_S, and the task communication rate μ_C are both 16 tasks/time–unit. System topology is a 16–node hypercube network and load update period, T_u, is 0.5 time–units. In Figure 6, system load ϱ is 0.5 (with λ = 0.5 and μ_E = 1). Figure 7 differs from Figure 6 in that the system load is increased from 0.5 to 0.8. From these figures, we observe the following:

- The difference in the response time computed from the model and the response time observed from simulation is very small. For most of the cases, this difference is less than 1%. The worst case difference is 6.52%.
- At low loading conditions, *FAverage* performs well, whereas *PRandom* performs the worst of all. The difference in the performance of *FRandom* and *PRandom* is not significant, implying that for random algorithms, information exchange can be done either instantaneously or periodically with T_u = 0.5.
- The difference in the performance of *FMin* and *PMin* is not significant. Again, this implies that information update can be done by selecting either of the two principles.

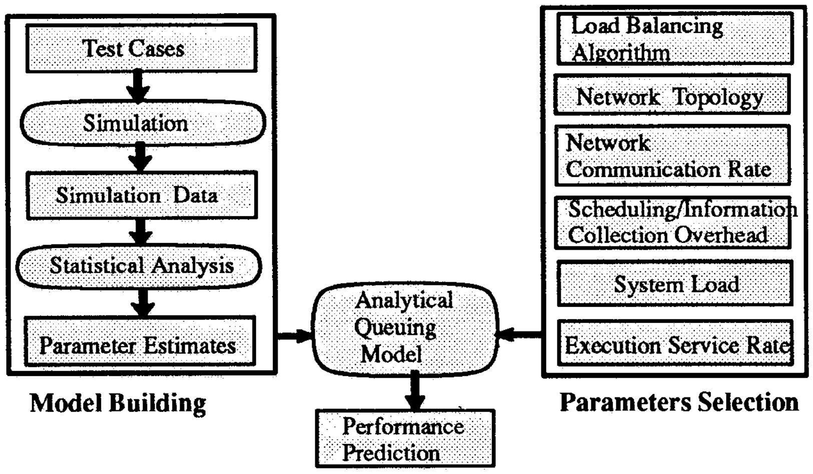


Figure 3: The complete Performance Prediction Model.

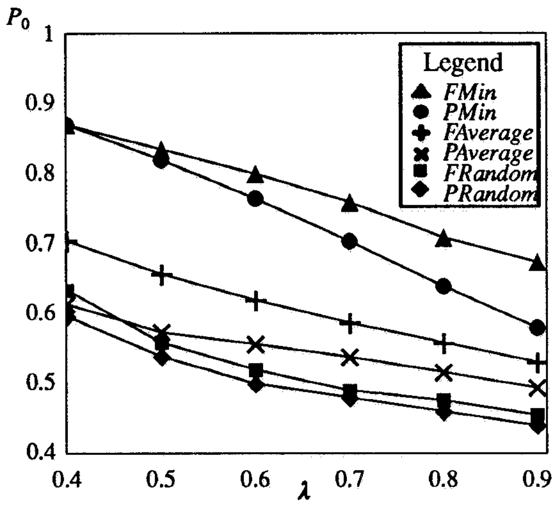


Figure 4: Variations in Probability P_0 versus system load for various load balancing strategies.

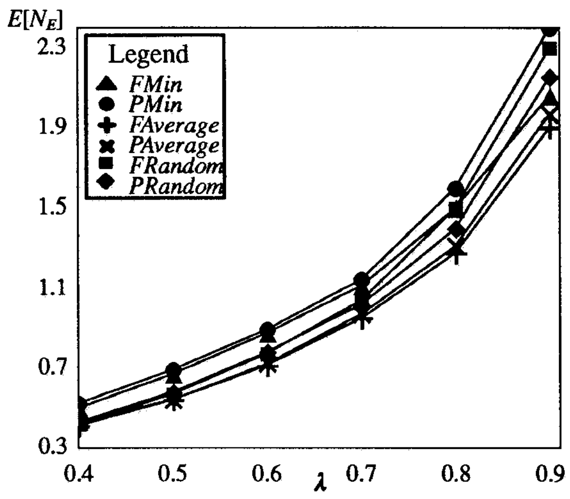


Figure 5: Variations in $E[N_E]$ length versus system load for various load balancing strategies.

This indicates that with T_u = 0.5, periodic update strategies perform as well as fresh information update strategies.

In order to check the validity of the proposed model for various parameters, we change μ_S and μ_C but keep the rest fixed. These results are shown in Figures 8 and 9. A high system load, equal to 0.8, is selected by first considering a fast communication network and slow task scheduling rate (μ_C = 16 tasks/time–units and μ_S = 8 tasks/time–units), and then considering a slow network and fast task scheduling rate with (μ_C = 8 tasks/time–units and μ_S = 16 tasks/time–units). Again, the model is shown to predict the average response time, which closely matches the response time produced by simulation. Further insights drawn from these figures are summarized below.

- We note that task scheduling time has greater impact on the average task response time than the task communication time. This is obvious because the average response time with a slow scheduling rate and high communication rate (Figure 8) is greater than the response time with a fast scheduling rate and slow communication rate (Figure 9). The observation is true for all strategies.

Next, we show two arbitrarily chosen sets of system parameters. In the first set, a 16–node folded hypercube with 5 links per node at a relatively low system load (0.6) is selected. The task communication rate and the task scheduling rate are both 12 tasks/time–unit and T_u is equal to 1.5 time–unit, which is relatively large. The results for this combination of parameters are shown in Figure 10 and are summarized below.

- The difference in the response time for the model and simulation is again very small.
- The periodic update strategies, *PMin* and *PAverage*, are outperformed by FMin and FAverage because of the larger value of T_u .

Comparison of response times predicted by the model and simulation for various strategies with various system parameters on a 16–node hypercube topology.

Legend: Model: ■ Simulation: ▢

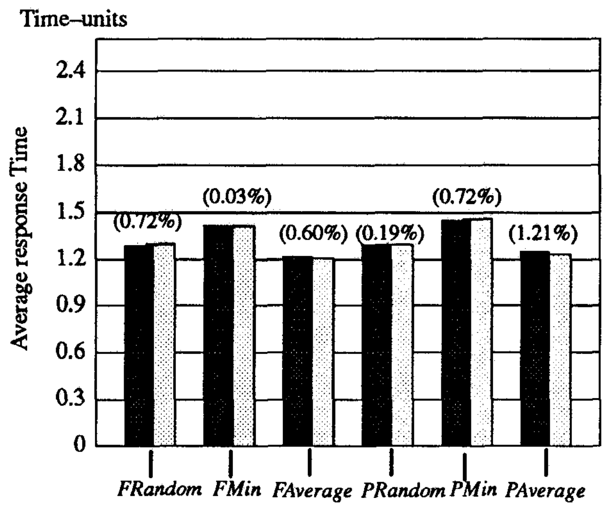


Figure 6: $\lambda = 0.5$, $\mu_C = 16$ task/time–unit, $\mu_S = 16$ task/time–unit, $T_u = 0.5$ time–units.

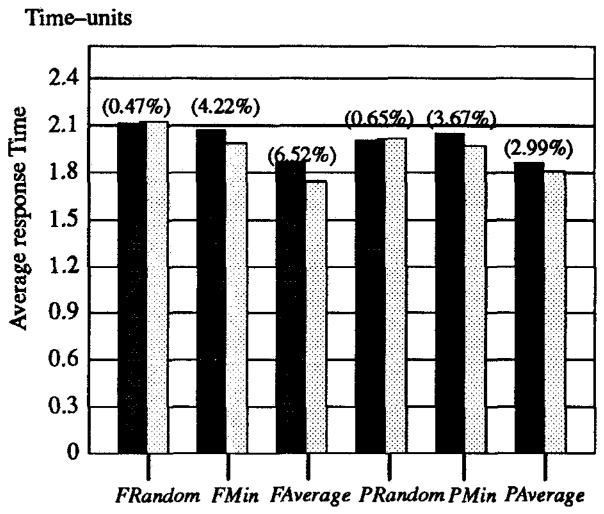


Figure 7: $\lambda = 0.8$, $\mu_C = 16$ task/time–unit, $\mu_S = 16$ task/time–unit, $T_u = 0.5$ time–units.

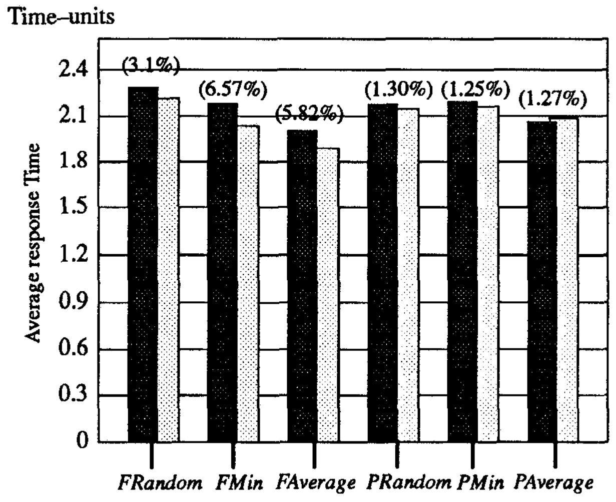


Figure 8: $\lambda = 0.8$, $\mu_C = 16$ task/time–unit, $\mu_S = 8$ task/time–unit, $T_u = 1.0$ time–units.

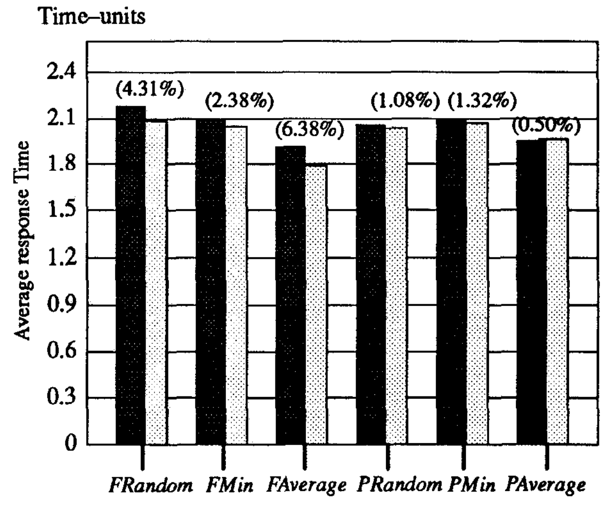


Figure 9: $\lambda = 0.8$, $\mu_C = 8$ task/time–unit, $\mu_S = 16$ task/time–unit, $T_u = 1.0$ time–units.

- On the other hand, FRandom and PRandom yield identical results by showing their insensitivity to the load update method.

In the second set, we have selected a 16–node ring network with medium system load equal to 0.7. Again, the response times predicted by the model match those produced by the simulation, as shown in Figure 11.

Up to this point, the performance of the model is compared with the same simulation test cases through which empirical data for statistical modeling was obtained. After characterizing P_0 and $E[N_E]$, the queuing model was used to compute the average response time and the results were compared with the same simulation results. Therefore, the comparison of the model with simulation has only revealed the correctness of the model. The validity of the proposed model is more strongly established as we obtain response time from the model and compare it with some additional

Comparison of response times predicted by the model and simulation for various strategies with various system parameters on different network topologies.

Legend: Model: ■ Simulation: □

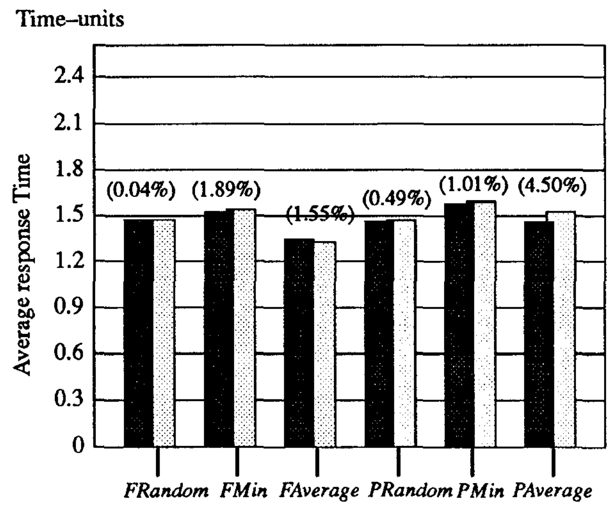


Figure 10: $\lambda = 0.6$, $\mu_C = 12$ task/time–unit, $\mu_S = 12$ task/time–unit, $T_u = 1.5$ time–units. Topology = 16–node Folded Hypercube.

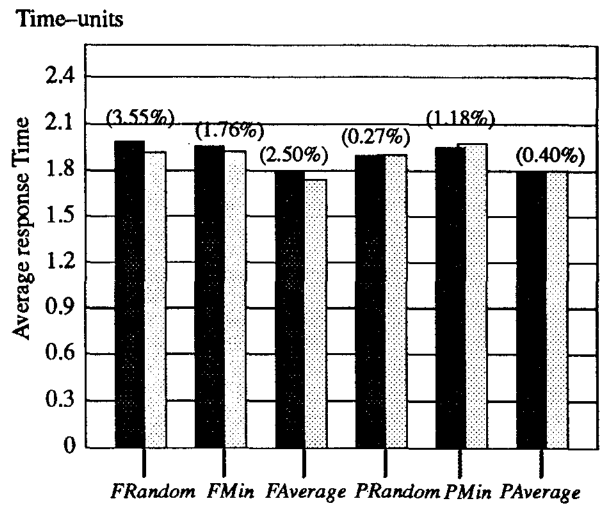


Figure 11: $\lambda = 0.7, \mu_C = 12$ task/time–unit, $\mu_S = 12$ task/time–unit, $T_u = 1.0$ time–units. Topology = 16–node Ring..

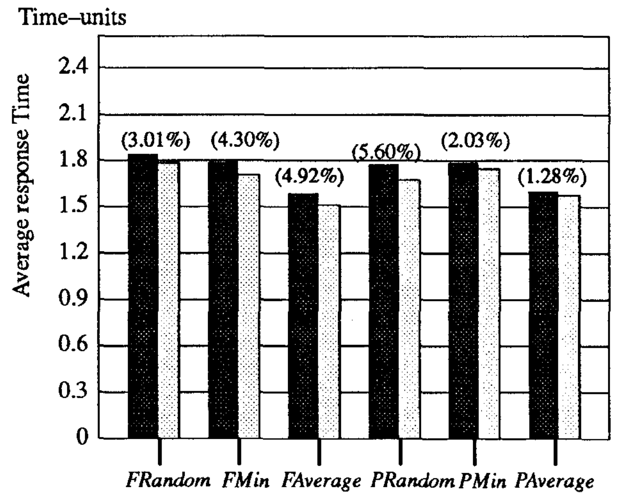


Figure 12: $\lambda = 0.7, \mu_C = 16$ task/time–unit, $\mu_S = 16$ task/time–unit, $T_u = 1.0$ time–units. Topology = 9–node Mesh.

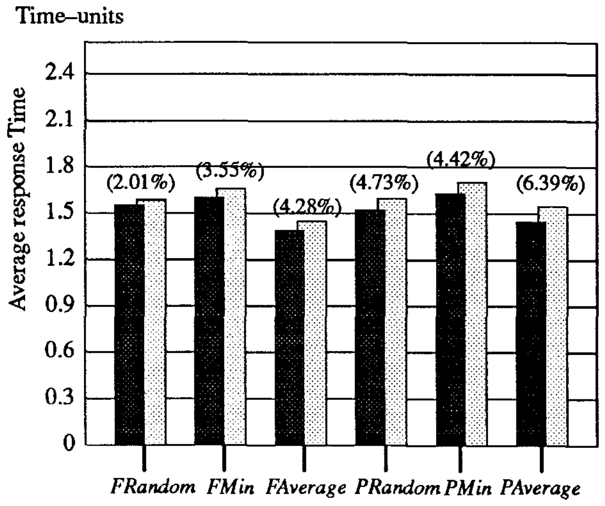


Figure 13: $\lambda = 0.7, \mu_C = 16$ task/time–unit, $\mu_S = 16$ task/time–unit, $T_u = 1.0$ time–units. Topology = 8–node Fully Conected.

simulation runs. The empirical data from these simulation runs has not been used for statistical modeling. The additional simulation runs include different network topologies with different parameters. The results of some combinations are shown in Figures 12 and 13. By examining these figures, we conclude the following.

- Again, the difference between simulation and model is small.
- *PAverage* performs as well as *FAverage*, if T_u is small.
- All nearest neighbor load balancing strategies perform better if the number of links per node are increased. This is because the probability that a node finds a suitable neighbor for task migration improves with the increase in the number of links.

- The difference in the performance of 'random' strategies and 'min' strategies is not very significant as compared to the difference in the performance of 'random' and 'averaging' strategies.
- Random algorithms can be used with periodic information updates for any network topology because they generate less message traffic. This is especially true for the fully connected network where *PRandom* performs as well as *FRandom.*
- If the actual scheduling time, $1/\mu_s$, for the random algorithm is less than that for 'min' algorithms, *PRandom* can be used instead of *FMin*, *PMin* or *FRandom.*

5. Concluding Remarks

In this paper, we have presented an approach for modeling the average task response time for distributed load balancing in multicomputer systems. With this approach, we are able to compare different load balancing schemes on a unified basis. We have shown that these strategies can be modeled by an open central server queuing network if the system is symmetric and homogeneous. We believe that any sender-initiated load balancing strategy can be modeled by this queuing network. With examples from a wide range of system parameters, it is shown that the average task response time predicted through the proposed model closely matches the response time obtained via simulation. This approach can be useful for analyzing and tuning an existing system, and evaluating newly proposed strategies.

Acknowledgements

The authors express their gratitude to anonymous referees whose comments improved the quality of this paper. Thanks are also due to NPAC, Syracuse University, in particular the system administration staff, for making Multimax available during the last hours.

References

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

[2] Raymond M. Bryant and Raphael A. Finkel, "A Stable Distributed Scheduling Algorithm," in *Proc. of 2nd Int'l. Conf. on Distributed Computing Systems*, 1981, pp. 314–323.

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

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

[5] G. C. Fox, A. Kolawa and R. Williams, "The Implementation of a Dynamic Load Balancer, "in *Proc. of SIAM Hypercube Multiprocessors Conf.*, 1987, pp. 114–121.

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

[7] A. Ghafoor, T. Bashkow and Imran Ghafoor, "Bisectional Fault–Tolerant Communication Architecture for Supercomputer Systems, " *IEEE Trans. on Computers*,vol. 38, no. 10, pp. 1425–1446, October 1989.

[8] D. C. Grundwald, B. A. Nazief and D. 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] A. Ha'c and T. J. Johnson, "Sensitivity Study of the Load Balancing Algorithm in a Distributed System," *Journal of Parallel and Distributed Computing*, October 1990, pp. 85–89.

[10] L. V. Kalé, "Comparing the performance of two dynamic load distribution methods," Proceedings of *Int'l. Conf. on Parallel Processing*, 1988, pp. 8–12.

[11] F. C. H. Lin and R. M. Keller, "Gradient Model: A demand Driven Load Balancing Scheme," in *Proc. of 6–th Int'l Conf. on Distributed Computing Systems*, 1986, pp. 329–336.

[12] M. L. and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," in *Proc. of ACM Computer Network Performance Symposium*, April 1982, pp. 47–55.

[13] L. M. Ni, C. Xu and T. B. Gendreau, "A Distributed Drafting Algorithm for Load Balancing," *IEEE Trans. on Software Eng.*, vol. SE–11, no. 10 Oct. 1985, pp. 1153–1161.

[14] L. M. Ni and Kai Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Trans. on Software Eng.*, vol. SE–11, May 1985, pp. 491–496.

[15] K. Ramamritham, J. A. Stankovic and Wei Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *IEEE Trans. on Computers*, vol. 38, no. 8, Aug. 1989, pp. 1110–1123.

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

[17] V. 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.

[18] K. G. Shin and Y. –C. Chang, "Load Sharing in Distributed Real–Time Systems with State–Change Broadcasts," *IEEE Trans. on Computers*, vol. 38, no. 8, Aug. 1989, pp. 1124–1142.

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

[20] J. 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.

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

[22] J. Xu and K. Hwang, "Heuristic Methods for Dynamic Load Balancing in a Message–Passing Supercomputer,"in *Proc. of Supercomputing '90*, November 1990, pp. 888–897.

[23] Y. Wang and R. J. T. Morris, "Load Sharing in Distributed Systems," in *IEEE Trans. on Computers, C–34 no. 3*, March 1985, pp. 204–217.