1991

Performance Prediction for Distributed Load Balancing on Multicomputer Systems

Ishfaq Ahmed  
*Syracuse University*

Arif Ghafoor  
*Purdue University*

Kishan Mehrotra  
*Syracuse University*, mehrotra@syr.edu

Follow this and additional works at: https://surface.syr.edu/lcsmith_other

Part of the Computer Sciences Commons

**Recommended Citation**

https://surface.syr.edu/lcsmith_other/21

This Article is brought to you for free and open access by the College of Engineering and Computer Science at SURFACE. It has been accepted for inclusion in College of Engineering and Computer Science - Former Departments, Centers, Institutes and Projects by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.
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 decay 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 multiprocessor 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 $\lambda$ 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 $1/\mu_r$ 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 $1/\mu_r$ 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 1: 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 its own load information to all its neighbors periodically. The time period, $T$, 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 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, $k_0, k_1, \ldots, k_L$, 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 $j$ ($j = 0, 1, \ldots, L$) is given by the product:

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

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

$$ p_j(k_j) = (1 - q_j)^{k_j} q_j^j $$

![Figure 1: Distributed load balancing represented by open central server model.](image)
For the $j$-th component, the average utilization, $Q_j$, is equal to $\frac{\lambda_j}{\mu_j}$. 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 $P_i$. 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:

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

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:

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

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

$$E[R_s] = \frac{1}{\lambda} \sum_{j=0}^{L} \frac{Q_j}{1 - Q_j} \frac{1}{P_{E[j]}},$$

$$= \frac{1}{\lambda} \frac{1}{(P_{E[0]}} + \sum_{j=1}^{L} \frac{P_j}{(P_{E[j]}} + \frac{P_j}{(P_{E[j]}}) \frac{1}{\lambda P_j} / (\mu P_s)$$

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, $\mu_0$ and $\mu_1$'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 $\lambda, \mu_0, \mu_1, \mu_2, T_0$ 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 $\lambda, \mu_0, \mu_1, \mu_2, \mu_3, T_0$, 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, \( \lambda \) was varied from 0.3 to 0.9 tasks per time-unit, \( \mu_c \) was varied from 8 to 16 tasks per time-unit and \( \mu_e \) was varied from 8 to 16 task per time-unit. The task execution rate, \( \mu_e \), 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, \( \mu_e \). In other words, when \( \mu_e \) is 10 tasks/time-unit and \( \mu_e \) 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 \( \lambda \), \( \mu_c \), \( \mu_e \), \( T_u \) 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 \( \alpha \) and coefficients, \( \beta \)'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 \( \mu_c \) and \( \mu_e \) 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

---

**Table I: Estimation for \( P_0 \) and its sensitivity versus system parameters.**

<table>
<thead>
<tr>
<th>Strategy</th>
<th>R–Square</th>
<th>System Parameter</th>
<th>Coefficient Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRandom</td>
<td>0.9277</td>
<td>( \alpha )</td>
<td>1.72220</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_c )</td>
<td>-0.15421</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_s )</td>
<td>0.00116 +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>-1.32043</td>
</tr>
<tr>
<td>FMin</td>
<td>0.9505</td>
<td>( \alpha )</td>
<td>3.39618</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_c )</td>
<td>0.00881</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_s )</td>
<td>0.00841</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>-1.04726</td>
</tr>
<tr>
<td>FAverage</td>
<td>0.8668</td>
<td>( \alpha )</td>
<td>1.48038</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_c )</td>
<td>0.00839</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_s )</td>
<td>0.01214</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>-1.10472</td>
</tr>
<tr>
<td>PPandom</td>
<td>0.9356</td>
<td>( \alpha )</td>
<td>1.63440</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( T_u )</td>
<td>-0.13364</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_c )</td>
<td>-0.13337</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_s )</td>
<td>-0.00214 +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>-0.00395</td>
</tr>
<tr>
<td>PMin</td>
<td>0.9683</td>
<td>( \alpha )</td>
<td>4.06852</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( T_u )</td>
<td>-0.21302</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_c )</td>
<td>-0.10590</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_s )</td>
<td>0.00013 +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>-3.49994</td>
</tr>
<tr>
<td>PAverage</td>
<td>0.9038</td>
<td>( \alpha )</td>
<td>2.16996</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( T_u )</td>
<td>-0.42800</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_c )</td>
<td>-0.14715</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_s )</td>
<td>-0.00146 +</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>-1.46303</td>
</tr>
</tbody>
</table>

Note: All estimates of model parameters are statistically significant, except slightly significant(\( + \)), and not significant(\( \dag \)).
\[ P_0 = 1 + e^{-\left(\alpha + \beta_1 \text{links} + \beta_2 \mu_c + \beta_3 \mu_s + \beta_4 \lambda + \beta_5 T_s\right)} \]  

Equation (I)

\[ E[N_e] = \exp\left(\alpha + \beta_1 \text{links} + \beta_2 \mu_c + \beta_3 \mu_s + \beta_4 \lambda + \beta_5 T_s\right) \]  

Equation (II)

Table II: Estimation for \( E[N_e] \) and its sensitivity versus system parameters.

<table>
<thead>
<tr>
<th>Strategy</th>
<th>R-Square</th>
<th>Parameter</th>
<th>Parameter Estimate</th>
</tr>
</thead>
<tbody>
<tr>
<td>FRandom</td>
<td>0.9931</td>
<td>( \alpha )</td>
<td>–1.96055</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>–0.06211</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_c )</td>
<td>3.30085</td>
</tr>
<tr>
<td>FMin</td>
<td>0.9926</td>
<td>( \alpha )</td>
<td>–1.67735</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>–0.03932</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_s )</td>
<td>–2.86089</td>
</tr>
<tr>
<td>FAverage</td>
<td>0.9945</td>
<td>( \alpha )</td>
<td>–1.92121</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>–0.05420</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_c )</td>
<td>3.07851</td>
</tr>
<tr>
<td>PRandom</td>
<td>0.9981</td>
<td>( \alpha )</td>
<td>–1.86021</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>–0.02244</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_s )</td>
<td>–0.05787</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( T_u )</td>
<td>3.08250</td>
</tr>
<tr>
<td>PMin</td>
<td>0.9953</td>
<td>( \alpha )</td>
<td>–1.59469</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>0.01106</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_c )</td>
<td>–0.03587</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_s )</td>
<td>2.70488</td>
</tr>
<tr>
<td>PAverage</td>
<td>0.9984</td>
<td>( \alpha )</td>
<td>–1.85252</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \lambda )</td>
<td>–0.00950</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_c )</td>
<td>–0.05200</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( \mu_s )</td>
<td>2.94949</td>
</tr>
</tbody>
</table>

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 \( F\text{Average} \) followed by \( P\text{Average} \), and \( P\text{Min} \) 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 \( \lambda \), \( \mu_c \), \( \mu_s \), \( T_u \) 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 ±7%. Since all results cannot be provided within this paper, we present only those results with an 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, \( \mu_s \), and the task communication rate \( \mu_c \) 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 \( \rho \) is 0.5 (with \( \lambda = 0.5 \) and \( \mu_s = 1 \)). 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, \( F\text{Average} \) performs well, whereas \( P\text{Random} \) performs the worst of all. The difference in the performance of \( F\text{Random} \) and \( P\text{Random} \) 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 \( F\text{Min} \) and \( P\text{Min} \) is not significant. Again, this implies that information update can be done by selecting either of the two principles.
This indicates that with $T_s = 0.5$, 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 $\mu_s$ and $\mu_C$ 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 ($\mu_C = 16 \text{ tasks/time-units}$ and $\mu_s = 8 \text{ tasks/time-units}$), and then considering a slow network and fast task scheduling rate with ($\mu_C = 8 \text{ tasks/time-units}$ and $\mu_s = 16 \text{ tasks/time-units}$). 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).

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 $T_s$ 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 $T_s$. 

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.
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: □

![Graph](image)

**Figure 6:** $\lambda = 0.5$, $\mu_c = 16$ task/time-unit, $\mu_s = 16$ task/time-unit, $T_s = 0.5$ time-units.

**Figure 7:** $\lambda = 0.8$, $\mu_c = 16$ task/time-unit, $\mu_s = 16$ task/time-unit, $T_s = 0.5$ time-units.

**Figure 8:** $\lambda = 0.8$, $\mu_c = 16$ task/time-unit, $\mu_s = 8$ task/time-unit, $T_s = 1.0$ time-units.

**Figure 9:** $\lambda = 0.8$, $\mu_c = 8$ task/time-unit, $\mu_s = 16$ task/time-unit, $T_s = 1.0$ 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 $E[N_q]$, 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: \( \lambda = 0.6, \mu_C = 12 \text{ task/time-unit}, \mu_S = 12 \text{ task/time-unit}, T_s = 1.5 \text{ time-units.} \)
Topology = 16-node Folded Hypercube.

Figure 11: \( \lambda = 0.7, \mu_C = 12 \text{ task/time-unit}, \mu_S = 12 \text{ task/time-unit}, T_s = 1.0 \text{ time-units.} \)
Topology = 16-node Ring.

Figure 12: \( \lambda = 0.7, \mu_C = 16 \text{ task/time-unit}, \mu_S = 16 \text{ task/time-unit}, T_s = 1.0 \text{ time-units.} \)
Topology = 9-node Mesh.

Figure 13: \( \lambda = 0.7, \mu_C = 16 \text{ task/time-unit}, \mu_S = 16 \text{ task/time-unit}, T_s = 1.0 \text{ time-units.} \)
Topology = 8-node Fully Connected.

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.

- \( P_{\text{Average}} \) performs as well as \( F_{\text{Average}} \), if \( T_s \) 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 Random performs as well as Random.
• If the actual scheduling time, $T$, for the random algorithm is less than that for 'min' algorithms, Random 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