A Distributed Scheduling Simulation

David Baker, Russell Haddleton, and Kevin Wika

University of Virginia

Abstract

The paper presents a distributed scheduling simulator designed for the analysis of distributed scheduling algorithms. The purpose of the paper is to present the scheduler and show its ability to test various scheduling policies. Although preliminary test results are shown, these results are used to show the effectiveness of the scheduler, not to compare scheduling policies. Using a simulation tool named SES Workbench as its base, the simulator has the flexibility to represent multiple types of network configurations. By supporting the ability to vary both the scheduling policy being used and the network configuration, the simulator is a valuable tool for testing the feasibility of a given scheduling policy on different types of distributed systems.

I. Introduction

An operating system can be viewed as a process which is responsible for the management and allocation of computer resources. One of the most important resources is the processor. The management of the processor is really a problem of scheduling tasks to execute in accordance with the goals established for the processor resource. There are several (possibly conflicting) goals including maximizing response time, maximizing throughput, increasing CPU utilization and giving priority to particular jobs or users. For a single CPU host, the scheduling problem is a fairly straightforward task and much work has been done in this area[1]. Scheduling is considerably more complicated for distributed systems. Difficulties occur because there is often no shared state, communication costs are usually higher, hosts can be heterogeneous, network and host faults must be considered, and in general it is not possible to arrive at an optimal solution to the problem. Researchers have tackled the problem using analytical approaches, simulations and algorithms implemented in real distributed systems. For our exploration of distributed scheduling, the simulation option is the best choice. It provides greater flexibility than an analytical approach and can give a good first approximation to a real system without many of the associated complexities.

In developing a simulation our goal was to produce a powerful yet flexible system for the evaluation of distributed scheduling algorithms. We attempted to meet these goals by using a professional simulation tool (SES/Workbench), by designing the system to be highly paramaterizable, by providing mechanisms useful for the implementation of a range of scheduling policies, and by providing a simple means of altering the model parameters. The system provides for simulation of a network with up to three sites with user-specified numbers of heterogeneous hosts at each site. Variable parameters include the number of sites, host counts and characteristics, local and remote communication costs, system load, host run queue lengths, and job types and distribution.

II. Background

As one would expect, considerable work has been done in the area of distributed scheduling. Casavant and Kuhl[2] provide a summary of some of the work and provide an extensive bibliography. One of the most important papers in the area is credited to Eagar and Lazowska[3]. Using an analytical model, they attempted to determine what types of transfer and location policies would be most effective. In particular, they were interested in comparing policies that used larger amounts of system state information to simpler policies that used little or no state information. What they discovered is that the simpler policies achieved performance that was often close to the ideal M/M/K case. More complex policies did not significantly improve performance because of their higher overhead and occasionally suffered from instability because they were using specific information that could be outdated by the time an action was taken. Others have confirmed the general principle of simple is better with heterogeneous systems[4]. However, with heterogeneous systems the results were shown to be much more sensitive to the values of parameters such as the threshold (the value used to determine whether a host is "full" or able to accept new jobs).
III. Description of experimental arrangement

A. SES/Workbench

SES/Workbench™ is a collection of tools designed to aid in the specification and evaluation of system designs[5]. Simulations in SES are implemented using the event list approach. The three main components of the package are SES/design which is used to graphically specify a model, SES/Sim which compiles the model to executable form and SES/Scope which provides a facility for animating the simulation of a system design.

The system model is constructed and parameterized using the graphical interface provided by SES/Design. The physical model is constructed from a collection of nodes and interconnecting arcs.

The nodes in SES/Design are connected by arcs which are responsible for carrying transactions. Each arc has parameters specifying what types of transactions it will accept and additional selection can be made based on data within a transaction (e.g., the number of the destination host is frequently used in this simulation). Both nodes and arcs are manipulated using tools provided by SES/Design.

Jobs, messages and in general any entity that moves through an SES/Workbench model is represented by a transaction. Transactions are data packets that have SES/Workbench declared information and can contain user declared data objects also. For example, jobs have the user declared objects memory size, processor service time, destination site, and destination host number. Transactions can be declared to be of different categories (types) and in this simulation transactions are in the category job or message.

SES/Design provides for the creation of abstractions using submodels. For example, the distributed scheduling simulation has a host submodel for each different type of host, a site submodel, and a network submodel. Once a submodel has been defined, it can subsequently be used in any other submodel using a reference node. Fig. 1 shows the host submodels referenced in a site submodel. Additionally, arrays of submodels can be created as a means of producing multiple instances of a submodel. This technique is used to make multiple instances of a host type.

When determining certain parameters such as interarrival time, processor service time and job communication delay, it is often convenient to generate the values using some probabilistic distribution. In the simulation, an exponential distribution is used for job interarrival times and a normal distribution is used to get the service time for a new job.

A set of pre-defined as well as user defined statistics can be collected using tools provided by SES/Design. The statistics are generated each time the simulation is run and are stored in a file. The statistics needed for the evaluation of the distributed scheduling policies are described in a later section.

B. Distributed system model

A distributed system can be modeled as a collection of sites connected by a communication network. The sites in turn are collections of possibly heterogeneous hosts which also communicate via a network. Fig. 1 shows a site with its communication network and five groups of hosts. (Each individual group is homogeneous.) We chose processor types that would have a wide range of processing power with the minimum being the IBM PS2 and the maximum represented by a Cray. The actual powers associated with the types are only estimates and in fact the names are used only to aid in remembering the host types. For the simulation, the important goal was modeling a network with multiple sites where each site could have diverse CPU resources.
In addition to the CPU power, the other host characteristics that can be altered include the memory size, the local scheduling policy, and the number of processors associated with a host. These parameters are set in the module global declarations node or by entering the hosts via the graphical interface and directly adjusting the parameters (most of the other model parameters are changed more easily and without recompiling using the parameter file).

Fig. 2 Model of a host.

Jobs and request messages enter the host via the enter node shown on the left side of the diagram. The request messages are routed along the top arc where the necessary state information is placed in the message before it is returned to the probing host. Entering jobs travel the arc to the first user-defined node where the transfer policy is implemented. Jobs processed locally are directed to the bottom arc where memory is allocated and the jobs are executed by the processor. The location policy is implemented by the rest of the model. A job to be transferred travels through the fork node where a probe transaction is created if the location policy uses messages. Newly created probes are given a direction, pass through a delay, and travel to the rest of the network via a reference node. When a probe returns, the state information is analyzed and the probe is either sent to another host or a transfer destination is set for the job. Once a destination is set, the job leaves the join node, passes through a delay node, and returns to the network to be routed to a new host. If a job cannot be transferred it is directed to the local CPU for processing.

The hosts within a site and the sites themselves are connected by a fairly simple communication network shown in Figs 1 and 3. Fig. 3 shows the highest level view of the system with the sites in the middle of the network. To the left of the network is the job producer and on the right side is the job consumer. Both of these will be described in detail below. The network is very simple and is designed solely to transport jobs and messages from one host to another, from the job producer, and to the job consumer. The network itself is not intended to model any particular network configuration or performance. Jobs being delivered to the hosts leave the job producer and travel to the fork in the network where they are directed along one of the arcs to a particular site. Direction of jobs at junctions is accomplished by setting accept conditions within the arcs based on the site, type, and number of the destination host. Once inside a site, a job or message is directed to a host type and finally to the host with the correct number.

When looking at the model, the presence of multiple hosts of a given type is not as obvious as the presence of host types within a site. This is because SES/Workbench allows the creation of arrays of submodels, so that it is possible to have multiple instances without having to physically replicate the hosts. On the exit side of the hosts, the network must be able to accept completed jobs, jobs that are being transferred to other hosts and messages that are being sent to other hosts. Completed jobs travel through the network and are directed to the job consumer. Jobs and mes-
Fig. 3 Representation of the network.

Figures destined for other hosts travel to the outer network and then travel the lower arc (see Fig. 3) back to the central point to the left of the sites where they are shuttled to the appropriate site and host.

As mentioned above, the network is not intended to model any real network either in topology or performance. The network simply provides a path for travel, and jobs and messages traveling the network arcs experience no delay. However, delays can be effected within the hosts themselves using an SES/Workbench delay node. These delays can be specified by the user with two common options, one being a constant delay and one a delay that is a function of message size and destination. The advantage of this approach is that one can establish a simple static structure for delivery of transactions and then model network performance using a description of the network. In the simulation as presently implemented a constant delay is used for messages. Different values are used for messages sent within a site and those sent to another site. Job transfer delays are calculated as a certain fraction of the job's requested service time and once again are different for intra and inter site transfers. The service request time was used to calculate the delay because that was the approach used by Eagar and Lazowska and it seemed as reasonable as some of the other alternatives. However, arguments could also be made for using a constant time (such as in the case where job transfer consists of moving a name) or calculating the delay as a function of the job's storage space requirements.

The job producer is responsible for the creation of jobs, the selection of a destination host, and the assignment of appropriate values for job size (memory requirements), required processor time, host preference if any, and a value indicating CPU effectiveness. Host preference is a real number indicating the likelihood that a job is executed on a specified host type (e.g., an image processing application might have a high affinity for a host with special purpose hardware). Similarly, CPU effectiveness is a value which indicates how effectively a given host processes a job type. An example of its use would be decreasing the execution time of a vectorizable job that runs on a vector processor.

The job producer consists of the source node and the user defined node shown at the left of Fig. 3. The source node originates transactions and the user defined node sets the destination and resource requirements. A table is used to determine the types of jobs that will be created and their characteristics. The table consists of a list of job types and associated with each type is a mean memory requirement, mean processor service time, a standard deviation for the service request time, host preference if any, a probability which specifies the likelihood of a job of the given type being created at any one time, and a replication factor which indicates how many copies of the job should be produced when the job type is selected. The mean values are used in probabilistic functions to compute actual processor and memory requirements. This table is stored in a file and is read in when the simulation is initialized. The advantage of the table driven approach to job production is that a user can specify a custom mix of jobs based on the table entries. Job types are generated randomly with a distribution that is calculated from the probabilities specified in the table. The determination of a destination takes into account the possible host preference, the number of sites and number and type of machines at a site.

In addition to specifying the job types the user can also specify a particular system load, K, which is an important variable when testing a scheduling algorithm. The system load parameter is really a goal that the simulation attempts to achieve. It cannot meet it exactly because the exact job mix and processor service times are not known. However, statistics gathered during the operation of the simulation indicate that actual system load is usually within 1 percent of the goal. System load, K, is defined as the ratio of total processor time required by the jobs to the total processor resources of the system P.
The variable that can be adjusted to achieve a particular system load is the interarrival time for jobs. The interarrival time is calculated as follows. First, the total processing power, $P$, of the system is determined by taking the total number of hosts of each type and multiplying by their respective CPU power. Summing the total powers for each type gives the total power of the system. Next, the expected processor service time, $S$, for an average job is calculated using the mean service time, $S_i$, for each job type and the associated probability, $p_i$, as follows (both $S_i$ and $p_i$ need to be calculated to account for the replication factor possibly having a value other than 1).

$$ S = \sum_{\text{job types}} S_i \cdot p_i $$

Using $S$, the interarrival time $t_i$ is calculated as follows

$$ K = \frac{S}{P} $$

Solving for $t_i$ gives

$$ t_i = \frac{S}{K \cdot P} $$

This is the interarrival time that is subsequently used by the job producer (the simulation actually uses this interarrival time as the mean for an exponential distribution since that is probably more realistic than a completely fixed interarrival time). It is recognized that at any given time $K$ will not likely not be achieved; however, over a sufficiently long period of time the average system load should approach $K$. One note is that since the simulation starts with zero load, a certain start-up time will be required before the actual load on the system approaches the requested system load. The simulation takes this into account and uses a warm-up period provided by SES/Workbench to approach the system load before data is actually gathered.

The job consumer is simply the destination for all completed jobs. It is present so that transactions terminate at a central location, which is particularly useful for the collection of statistics (described below) related to jobs. Messages do not terminate at the job producer, but instead return to the originating host and are absorbed.

C. Statistics

As mentioned previously SES/Workbench provides tools for collection of statistical data. To evaluate our distributed scheduling policies we collected statistics on the number of jobs processed, the number of messages sent, job response time, job interarrival times, the number of probes and the number of job transfers. For the latter three categories, SES/Workbench maintains a record of the minimum and maximum values and calculates the mean, variance and standard deviation. One discovery related to the collection of statistics is that long jobs that are not completed when the simulation time expires are not included in the statistics. The result is that response times can be artificially low. For example, one can improve response time by scheduling a very large job on a slow host knowing that it likely won’t complete and therefore won’t affect the statistics. This can result in some very counter-intuitive results. One way to minimize the effect is to increase the simulation time so that large jobs are more likely to complete and even if some jobs don’t finish, they will have less impact on the overall results.

IV. Results and Testing

A. Policies Implemented

We have used SES/Workbench and the mechanisms described above to simulate several different distributed scheduling algorithms. The algorithms are all based on the run queue threshold transfer policy which has been successfully used in several algorithms in the literature[3,6]. The location policies presently implemented are random, shortest queue length and threshold which are all sender initiated, distributed policies predicated on the fact that migration is not supported.

The threshold transfer policy is a very simple policy where the decision to execute a job locally or transfer it is based solely on the state of the local host and the number of times the job may have been previously transferred. The length of the ready-to-run queue is compared to a predetermined threshold. If the queue length is less than the threshold or the number of transfers does exceed some maximum value, the job is executed locally, otherwise it proceeds to the location mechanism where an attempt is made to transfer it. The threshold value and the maximum number of transfers are specified in the parameter input file with a different threshold for each type of host. When the number of transfers exceeds the maximum for a site, the last host to receive the job must execute it locally unless there exists another site to which the job can be transferred (provided the number of total transfers has not been exceeded).

The random policy is a non-adaptive technique that functions just as its name would indicate. When a decision is made to transfer a task, a new destination site, host type and specific host are selected using the mechanisms described above. The job is then sent on to the new host which repeats the process. When a network has several sites, the job is transferred a given number of times within
the site, then it is transferred to another site where it may be transferred a set number of times and finally, if it has still not been accepted it is transferred to a third site. Once a job leaves a site it is never returned to that site. The number of transfers at the first and second site and the total number of transfers are specified in the parameter file.

The shortest queue length location policy is an adaptive policy which uses messages (probes) to determine the state of a specified number of hosts. When the decision is made to transfer a job, the local host sends out several probes (the exact number is specified in the parameter file) to other hosts. Once again, the destination host is determined using the location mechanisms. Each host that receives a probe compares its current ready-to-run queue length to its threshold length and returns this ratio in a message to the probing host. A ratio is used because the hosts can have different queue size values and those with higher queue size values should receive a greater proportion of the jobs. If just the ready-to-run queue length were used, all hosts would be treated the same regardless of their maximum queue size. As the host receives the replies, it keeps a record of the smallest ratio. The job is subsequently transferred to the host with the smallest ratio provided the ratio is less than 1 (i.e. the queue length is less than the threshold). If the ratio is greater than or equal to 1 and the number of site transfers is less than a specified value, the job will be transferred to another site. When the maximum number of site transfers is reached and none of the probed hosts has a ratio less than 1, the job is executed on the last host to receive it. The number of probes and the number of site transfers are entered in the parameter file.

The third location policy is the threshold policy, which is an adaptive policy similar to shortest queue length. (It is important to note that the name 'threshold' is used to refer to both the transfer policy used in combination with all the location policies and to refer to the specific location policy.) However, instead of waiting for all the probes to return with their state information, the threshold policy transfers the job as soon as a host is found with a ready-to-run queue length less than its threshold. As with shortest queue length, if a suitable host is not found within a set number of probes and the maximum number of site transfers has not been exceeded the job is sent to another site.

B. Testing

The model has been tested extensively during development. In addition, since the completion of the system, tests have been conducted to verify correct operation. Most of these tests have taken the form of acceptance checks, where a parameter is altered and the behavior of the model is observed. There are several cases where an intuition-based test of this variety can be applied.

The host affinity option was tested in this manner using a network of 20 small hosts with a power of 1 unit each and two more powerful hosts with a power of 20 units each. Two types of jobs were produced for the system. One with a mean service time of 1 was assigned a weight of 20 and a second type with a mean service time of 15 was given a
weight of 2. The weights result in a small job being produced with probability $\frac{20}{22} = 0.91$. In the test the affinity of the larger jobs for the more powerful hosts was varied. The expected result was that a higher affinity for the bigger hosts would result in a faster response time because the big jobs would be more likely to execute on a faster host, thereby making better use of the power of the large hosts. Fig. 4 shows a plot comparing response time to the value of the affinity of a large job for the more powerful host. There is an improved response time with the higher affinities which is expected. The three curves each represent a different CPU effectiveness for the smaller hosts executing big jobs. As expected the lower CPU effectiveness values resulted in higher response times especially when the affinity was low resulting in more big jobs executing on small hosts. This also served as a simple check of the effectiveness of the table guided job producer.

C. Verification of Eagar and Lazowzka

The paper by Eagar and Lazowska[3] provides a wonderful opportunity to verify the accuracy of our simulation and attempt to verify a portion of their work at the same time. In their paper they provided sufficient detail to closely approximate the assumptions they made. Additionally, since we used the same basic transfer and location policies we should expect very similar results.

We were particularly interested in determining the response time versus system load for the three policies and the limiting case of M/M/1 since this was the major result of their paper. The graph in Fig. 5 shows the results of our tests. The general shape of the plots is very similar to Eagar and Lazowska’s original graph, however we found that our response times were faster for all of the policies. One reason for this is probably due to the fact that they considered communication delays to be a processor cost while we chose to delay the message in the communication system. As a result, our processors are unaffected by the transfer of jobs and can continue to execute jobs in the ready-to-run queue. However, this does not explain the faster response time in the M/M/1 case since no jobs are being transferred in this case. In spite of the differences, a comparison of the policies would still produce the same conclusions: random is surprisingly effective, threshold is a significant improvement over random under heavy system load and shortest queue length does not improve much on threshold in spite of the fact that more information is gathered.

To further investigate the difference between communication delays as processor costs and as a simple network delay, the model was altered to implement communication delays as processor costs. The results for the homogeneous system indicated that there was not a significant difference between the two approaches. Therefore, because the original model was less complex it was retained. An additional reason for favoring the initial model is that it really models each processor as having a communications co-processor. Although, this may not be a reasonable assumption for smaller hosts (e.g., PCs and workstations) it is probably not unreasonable for the large and medium size hosts.

![Fig. 5. Response time versus system load](image)
Fig. 6 gives the results of varying the threshold for the threshold transfer and location policies. As Eagar and Lazowska found, a threshold value of 1 gives the best results until the system load surpasses 70%. Beyond this point a maximum of two probes gives the best performance until the system load becomes very high (> 95%).

D. Results for a Heterogeneous Distributed System

As a test of the effectiveness of the simulation in modelling a larger, heterogeneous, distributed system, a network consisting of three separate sites was set-up. One site had 20 PC's (1), the second had 2 IBM 3090's (160) and 15 Sun 4's (10), and the third 15 PC's, two VAX 9000's (40) and one Cray (1150). The numbers in parentheses give the relative powers for the five types of hosts.

The first test examines the effect of the selection weights on the response time. Table 1 shows the data with the intuitive result that the best response times were achieved when the more powerful hosts had the higher selection weights. In fact, the best results occurred when the selection weights closely matched the machine powers. It is reasonable that if the larger hosts receive more jobs there will be greater load sharing and hence improved response time. The table also shows that the improved response time is also a consequence of a reduced number of probes and transfers.

Tables 2 shows the effect of changing the number of probes on the response time of the threshold policy. The results for shortest queue were similar with both policies favoring smaller numbers of probes. The most effective combinations were those which had no more than two transfers at any given site. Higher numbers of probes resulted in poorer performance and a reduced return for the probing effort.

The number of variables in the hosts, job producer, network, and the policies make it difficult to single out the effects of any one parameter on the overall scheduling performance. However, to test the three policies we used some of the above results and opted to run a test of response time versus system load similar to that performed by Eagar and Lazowska. A network configuration using three sites and the same hosts as above was used for the experiment. The selection weights used were the same as the machine powers and the number of probes/transfers at each site was set at two. Communication delays were 0.001 and 0.015 for message transferred on and off site respectively and were 5% and 10% of the service request times for on and off site job transfers. Three types of jobs were created with the first having a mean size of 1, a probability of being produced of 0.75, and affinities of 1 for all hosts. The second job type had a mean size of 50, a probability of being produced of 0.20, and an affinity of 10 for IBM 3090's and 1 for all other hosts. Finally, the third job type had a mean size of 1000, a 0.05 probability of being created, and an affinity of 80 for the Cray and 1 for all other host types. The CPU effectiveness for all of the job types was set to 1 for each type of host.

Fig. 7 shows the response time versus system load using the parameters described above. The graph also shows the...
Table 1. Selection weight test using threshold policy.

<table>
<thead>
<tr>
<th>Number of Probes at Site: 1, 2, 3</th>
<th>Mean Response Time</th>
<th>Mean Transfers</th>
<th>Mean Site Transfers</th>
<th>Mean Probes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 1, 1</td>
<td>5.342</td>
<td>0.233</td>
<td>0.464</td>
<td>0.557</td>
</tr>
<tr>
<td>2, 1, 1</td>
<td>5.958</td>
<td>0.290</td>
<td>0.448</td>
<td>0.968</td>
</tr>
<tr>
<td>3, 1, 1</td>
<td>6.610</td>
<td>0.328</td>
<td>0.434</td>
<td>1.354</td>
</tr>
<tr>
<td>4, 1, 1</td>
<td>6.218</td>
<td>0.317</td>
<td>0.441</td>
<td>1.696</td>
</tr>
<tr>
<td>1, 2, 1</td>
<td>7.356</td>
<td>0.232</td>
<td>0.414</td>
<td>0.583</td>
</tr>
<tr>
<td>2, 2, 1</td>
<td>5.807</td>
<td>0.277</td>
<td>0.396</td>
<td>1.052</td>
</tr>
<tr>
<td>3, 2, 1</td>
<td>6.513</td>
<td>0.289</td>
<td>0.402</td>
<td>1.556</td>
</tr>
<tr>
<td>4, 2, 1</td>
<td>6.607</td>
<td>0.299</td>
<td>0.392</td>
<td>2.003</td>
</tr>
</tbody>
</table>

Table 2. Test of threshold location policy varying the number of probes.

<table>
<thead>
<tr>
<th>Selection Weights For Machines: 1, 2, 3, 4, 5</th>
<th>Mean Response Time</th>
<th>Mean Transfers</th>
<th>Mean Site Transfers</th>
<th>Mean Probes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 10, 5, 2, 4</td>
<td>20.21</td>
<td>0.219</td>
<td>0.268</td>
<td>1.125</td>
</tr>
<tr>
<td>5, 10, 5, 2, 4</td>
<td>19.57</td>
<td>0.199</td>
<td>0.337</td>
<td>1.378</td>
</tr>
<tr>
<td>5, 10, 40, 2, 4</td>
<td>15.73</td>
<td>0.196</td>
<td>0.257</td>
<td>1.088</td>
</tr>
<tr>
<td>1, 10, 40, 2, 4</td>
<td>15.01</td>
<td>0.102</td>
<td>0.214</td>
<td>0.764</td>
</tr>
<tr>
<td>1, 10, 60, 2, 4</td>
<td>12.04</td>
<td>0.075</td>
<td>0.241</td>
<td>0.839</td>
</tr>
<tr>
<td>1, 10, 60, 10, 4</td>
<td>17.24</td>
<td>0.193</td>
<td>0.240</td>
<td>0.900</td>
</tr>
<tr>
<td>1, 1, 50, 1, 1</td>
<td>15.11</td>
<td>0.190</td>
<td>0.253</td>
<td>0.988</td>
</tr>
<tr>
<td>1,160,1150,10,40</td>
<td>10.26</td>
<td>0.070</td>
<td>0.235</td>
<td>0.817</td>
</tr>
<tr>
<td>1,160,1150,1,40</td>
<td>9.43</td>
<td>0.064</td>
<td>0.224</td>
<td>0.783</td>
</tr>
</tbody>
</table>

Machine 1 = relative power of 1
Machine 2 = relative power of 160
Machine 3 = relative power of 1150
Machine 4 = relative power of 10
Machine 5 = relative power of 40

one limiting case of M/M/1 where no transfers occur. The M/M/K case is approximated by using shortest queue, setting the number of transfers to 6 at each site, and eliminating the communication delays. The approximation will not be totally correct because not all of the hosts in the network are simultaneously probed and the number of probes should be higher, but an array dimension limit was encountered. The graph shows that shortest queue length and threshold give similar results and are generally the best of the three policies except under very light loads. At all loads these two offer definite performance benefits over the M/M/1 limiting case. Random deteriorates as load increases and is actually worse than M/M/1 at 90% system load. In some instances the response time stays constant or is actually decreased as system load increases. This may be due to the fact that under heavier load, the smaller machines are more likely to transfer jobs thereby increasing the chance of finding one of the more powerful hosts.
V. Conclusion

As with any project, there are certainly many improvements and extensions to the work that we have done. A few that we are aware of include enabling jobs to generate other jobs, implementing other policies (e.g., receiver initiated), increasing the number of sites, attempting scheduling policies that utilize more information concerning hosts and job types, and making more realistic use of the memory resource. The other obvious work that is yet to be done is actually using the model to perform extensive evaluations of scheduling policies. We have built the tool and shown that it effectively simulates distributed scheduling, but have not had the opportunity to test a range of scheduling policies.

The goal of this work was to develop a simulation appropriate for the analysis of distributed scheduling algorithms. We feel that goal has been met for a subset of the possible scheduling algorithms. The simulation provides an effective, flexible tool for the evaluation of distributed scheduling policies. The modeling of a wide range of systems and policies is facilitated by the extent to which the simulation is parameterized (many of the parameters are read from a text file so they can be changed in O(text editor) time which is significantly less than O(compiler) time). Implementation of new scheduling policies would not be difficult given the support of SES/Workbench and the procedures that we have provided. The system that is in place should provide anyone interested in exploring distributed scheduling with a powerful tool for analyzing basic scheduling policies in a distributed, heterogeneous system.

VI. References