Measuring the Robustness of Resource Allocations in a Stochastic Dynamic Environment

Jay Smith[1,2], Luis D. Briceño[2], Anthony A. Maciejewski[2], Howard Jay Siegel[2,3], Timothy Renner[3], Vladimir Shestak[2], Joshua Ladd[4], Andrew Sutton[3], David Janovy[2], Sudha Govindasamy[2], Amin Alqudah[2], Rinku Dewri[3], Puneet Prakash[2] *

[1]IBM
6300 Diagonal Highway
Boulder, CO 80301
bigfun@us.ibm.com

Colorado State University
[2]Dept. of Electrical Engineering
[3]Dept. of Computer Science
[4]Mathematics Department
Fort Collins, CO 80523–1373 USA
{ldbricen,aam,hj}@colostate.edu

Abstract

Heterogeneous distributed computing systems often must operate in an environment where system parameters are subject to uncertainty. Robustness can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed. We present a methodology for quantifying the robustness of resource allocations in a dynamic environment where task execution times are stochastic. The methodology is evaluated through measuring the robustness of three different resource allocation heuristics within the context of a stochastic dynamic environment. A Bayesian regression model is fit to the combined results of the three heuristics to demonstrate the correlation between the stochastic robustness metric and the presented performance metric. The correlation results demonstrated the significant potential of the stochastic robustness metric to predict the relative performance of the three heuristics given a common objective function.

1. Introduction

Heterogeneous parallel and distributed computing is defined as the coordinated use of compute resources that have different capabilities to optimize system performance features. Often, heterogeneous, distributed computing systems must operate in an environment replete with uncertainty. Robustness in this context can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed [1]. We present the use of a stochastic robustness metric [14] to quantify the robustness of a resource allocation in a *dynamic* environment. This formulation of the stochastic robustness metric is used to predict the typical relative performance of three different resource allocation heuristics taken from the literature and adapted to the presented problem. A Bayesian regression model is fit to the combined results of the three heuristics to demonstrate the relationship between the stochastic robustness metric and the presented performance metric. The accuracy of the robustness predictions are then evaluated to determine their utility in predicting resource allocation heuristic performance in a dynamic environment.

The major contribution of this paper is a mathematical formulation for quantifying the robustness of a resource allocation in a stochastic dynamic environment and a methodology for applying the robustness formulation to predict heuristic performance. We demonstrate the proposed methodology by successfully predicting the relative performance of three heuristics taken from the literature and adapted to this environment. Finally, we present a detailed evaluation of the three heuristics using both the proposed robustness metric and a performance objective appropriate to the studied environment.

The environment considered in this research is that of a heterogeneous, distributed computing system designed to

*This research was supported by the NSF under Contract No: CNS-0615170, by the Colorado State University Center for Robustness in Computer Systems (funded by the Colorado Commission on Higher Education Technology Advancement Group through the Colorado Institute of Technology), and by the Colorado State University George T. Abell Endowment.

service a high volume web site of world-wide interest. The system being modeled was used to implement the 1998 World Cup web site [3] that processed more than 1.3 billion HTTP requests during the summer of 1998. The web site was provided to a world-wide audience by four heterogeneous, geographically dispersed systems—each with their own processing capacity and workload distribution techniques. This class of system is very challenging to implement but occurs surprisingly frequently. The World Cup football tournament is just one example of an event of world-wide appeal that necessitates web based coverage. Sites of this type are typically constructed for specific events and the volume of traffic is on the order of billions of requests in a period of only a few months or less. Other such events might include the Summer Olympics, the Winter Olympics, or the tour de France; all are reasonably expected to draw the attention of a world-wide audience in the billions.

This research developed resource allocation heuristics for a single location of a distributed system capable of processing a high volume of requests. Incoming requests were dispersed to one of four processing centers. Thus, each location was responsible for processing some fraction of the total traffic to the site. This work focused on a location comprising eight heterogeneous servers responsible for processing 45% of the overall traffic [2].

A task is defined to be a menu-driven HTTP request for data from the web site. Mapping tasks to machines in this distributed system is rather challenging as it must be done under uncertainty because the *exact* execution time required to process a task is not known *a priori*. However, past observations of task execution times can be used to construct a probability mass function (pmf) [17] that models the possible execution times for a given task. The pattern of task arrivals to the site was modeled after real traffic patterns observed by the 1998 World Cup web site [2].

In the next section, we present the details of the problem to be addressed. Section 3 defines a means for determining stochastic completion times for tasks in this system using the details of the problem statement. The definition of stochastic completion times is then used in Section 4 to derive a stochastic robustness metric that is subsequently used to predict heuristic performance. In Section 5, we present the heuristics that have been developed as part of this research. Details of the simulation environment are presented in Section 6 and the results of the heuristics are evaluated in Section 7. Section 8 presents an overview of the relevant related work. Section 9 concludes the paper.

2. Problem Statement

2.1. Introduction

The system studied in this research is an instance of a more general class of dynamic, heterogeneous computing (HC) system where task arrival times are not known in advance and exact task execution times are uncertain prior to their completion. All incoming tasks to the system are assumed to have been previously classified into one of C classes prior to their arrival. Each of the classes corresponds to a gross classification of the relative complexity of the request being processed. Each task class defines a set of pmfs, where each pmf describes the probability of all execution times for that class on a given machine within the HC suite. Further, all of the classes of tasks (HTTP requests) that the system may be asked to perform are known in advance, i.e., the web server has prior knowledge about what web pages it is providing.

Each arriving task has a relative deadline limiting the total time available to process each request. The relative deadline for each task class is assumed to have been established in advance. Because tasks in this environment are HTTP requests for data, made by a user of the website, it is assumed that if a task cannot be completed by its deadline then the request can be considered to be "timed out" and the user that submitted the original request will make the request again. Therefore, there is no benefit to completing tasks that miss their deadlines and, consequently, tasks that miss their deadlines will be discarded.

2.2. Performance Metric

In this environment, each incoming task has a hard deadline for its completion, i.e., failure to complete a task by its deadline will result in a penalty. To model the impact of missing a task deadline the resource allocation heuristic will be penalized by a constant factor of 1 for each task deadline that is missed. That is, for a given task i with deadline β_i^{max} let $comp(i)$ be the actual completion time of task i and define the cost to process a task i, denoted $cost(i)$, as follows

$$cost(i) = \begin{cases} 0, & \text{if } comp(i) \leq \beta_i^{max}; \\ 1, & \text{otherwise.} \end{cases} \quad (1)$$

The overall cost of a resource allocation is defined as the sum of the cost of each processed task. Define PT as the set of all tasks that are processed by the system, then the objective of a resource allocation in this environment can be expressed as,

$$\text{Minimize} \sum_{\forall i \in PT} cost(i). \quad (2)$$

Intuitively, resource allocations in this environment are expected to minimize the number of tasks that miss their deadlines.

2.3. Mapping Events

All of the resource allocation heuristics evaluated in this work operate in a pseudo-batch mode [10, 11]. In a pseudo-batch mode heuristic, all tasks that have not begun execution and are not next in line to begin execution can be considered for remapping when a mapping event occurs. Mapping events occur within the system whenever a new task arrives or an existing task completes.

3. Stochastic Task Completion Time

Although some tasks may belong to the same class, task execution times may still vary depending on the details of the data requested. For this reason, task execution times are modeled as random variables. In addition, it is reasonable to assume that each task execution time is independent because any single HTTP request can be satisfied without any need to process another HTTP request, i.e., each request is self-contained.

Let each task i belong to exactly one class c_i in the set of all task classifications C where membership in a class implies a specific random variable $\underline{T_{c_i j}}$ representing the execution time of that task class on machine j (one of the $\underline{M}$ machines in the HC suite). In this research, it is assumed that the probability distributions describing the random variable $T_{c_i j}$ were created from measurements of the response times of actual requests for data from the site. A typical method for creating such a distribution relies on a histogram estimator [17] that produces a discrete probability distribution known as a probability mass function. Define $\underline{f_{cj}}$ to be a unimodal pmf describing the execution time of tasks in class c on machine j. We consider only unimodal distributions of execution times to simplify the application of the stochastic robustness metric.

Determining the completion time for a machine j, and therefore the completion time of a particular task i, requires a means of combining the execution times for all tasks assigned to that machine. In a deterministic model of task execution times, the estimated execution times for all tasks assigned to machine j would be summed with the machine ready time to produce a completion time. A similar procedure is followed in the stochastic case as well. However, calculating stochastic completion times requires the summation of random variables as opposed to deterministic values. The summation of random variables given their pmfs can be found as the convolution of their corresponding pmfs [9].

Let $\underline{MQ(t)}$ be the set of all tasks that are either pending execution or are currently executing on any of the M machines in the HC suite at time t. To determine the completion time for a task i on machine j at time t, identify the subset of tasks in $MQ(t)$ that were mapped to machine j in advance of task i, denoted $\underline{MQ_{ij}(t)}$. The execution time pmfs for the pending tasks will be convolved [9] with the completion time distribution of the currently executing task and the execution time distribution for task i on machine j to produce the stochastic completion time pmf for task i on machine j.

The execution time pmf for the currently executing task on machine j requires some additional processing prior to its convolution with the pmfs of the pending tasks to create a completion time pmf. For example, if the currently executing task on machine j began execution at time t_j prior to time t, some of the impulse values of the pmf describing the completion time of the currently executing task may be in the past. Therefore, to accurately describe the completion time of task i at time t requires that these past impulses be removed from the pmf and the remaining distribution renormalized. After renormalization, the resulting distribution describes the completion time of the currently executing task at time t on machine j. To simplify notation, define an operator $\underline{GT(s, d)}$ that accepts a scalar s and a pmf d as input and returns a renormalized probability distribution where all impulse values of the returned distribution are greater than s. The completion time pmf of the currently executing task on machine j is determined by applying the GT operator to its completion time pmf, using the current time t. The resulting distribution is then convolved with the pmfs of the pending tasks on machine j and the execution time distribution of task i to produce the completion time pmf for task i on machine j at the current time t.

4. Stochastic Robustness Metric (SRM) for a Dynamic Mapper

4.1. Instantaneous SRM

Recall that the individual deadline of each task has been defined in advance; let β_i^{max} denote the deadline for the i^{th} task to arrive to the system. Let $f_{c_1 j}$ be the execution time pmf of the currently executing task on machine j. Order the members of $MQ_{ij}(t)$ according to their scheduled order of execution on machine j and let $f_{c_2 j}$ be the execution time pmf of the first pending task on machine j, with $f_{c_{|MQ_{ij}(t)|} j}$ as the execution time pmf of the last pending task on machine j that is ahead of task i.

Convolution of a scalar with a pmf has the effect of shifting the pmf by the value of the scalar and has no impact on the distribution of probabilities in the pmf. Therefore, if $MQ_{ij}(t) = \emptyset$, i.e., $MQ_{ij}(t)$ is empty, the completion time

distribution for task i on machine j is merely its execution time distribution f_{c_ij} shifted to the current time. The completion time distribution at time t for task i, denoted $\underline{F_i(t)}$ can be found as follows,

$$F_i(t) = \begin{cases} t * f_{c_ij}, & \text{if } MQ_{ij}(t) = \emptyset; \\ GT\{t, t_j * f_{c_1j}\} * f_{c_2j} * \cdots & \\ *f_{c_{|MQ_{ij}(t)|}} * f_{c_ij}, & \text{otherwise.} \end{cases} \tag{3}$$

Following from our prior work on robustness [14], the robustness of the finishing time for task i can be found as the probability that task i will finish before its deadline. This probability defines a local robustness characteristic, denoted $\underline{\psi_i(t)}$, and can be expressed as $\mathbb{P}\big[F_i(t) \leq \beta_i^{max}\big]$. The individual local robustness characteristics can then be combined to produce the stochastic robustness metric at time t, denoted $\psi(t)$, as follows,

$$\psi(t) = \prod_{\forall i \in MQ(t)} \Big(\mathbb{P}\big[F_i(t) \leq \beta_i^{max}\big]\Big). \tag{4}$$

This combination of local robustness characteristics defines an instantaneous measure of robustness (instantaneous SRM) for this resource allocation at a particular time t. Intuitively, the measure defines the probability that all tasks pending or currently executing at time t will meet their deadlines.

4.2. Dynamic SRM Value

The instantaneous measure of robustness is used as a basis for defining a single dynamic SRM value. To define that value, recall that a mapping event occurs whenever a task completes execution or a new task arrives at the system. An instantaneous SRM value is generated at each mapping event during the course of a simulation trial. These instantaneous SRM values are then combined to create a sample dynamic SRM value for the resource allocation heuristic. Given the relationship between the dynamic SRM value and the performance metric, the dynamic SRM value can be used to predict the relative performance of two resource allocation heuristics.

If a heuristic consistently maintains a high $\psi(t)$ value over some number of mapping events then there is a consistently low probability that tasks will miss their deadlines over that same period. Therefore, the average $\psi(t)$ value over a large enough number of mapping events should correlate with a consistently low probability that tasks will miss their deadlines. That is, heuristics that maintain a high average $\psi(t)$ value can reasonably be expected to produce a low cost. For this reason, the dynamic SRM value is defined as the average of the instantaneous SRM values found, at each mapping event, during a simulation trial.

4.3 Using the Dynamic SRM value

In a dynamic environment, the set of tasks being considered is constantly changing due to task arrivals and completions. Recall that to compute $\psi(t)$ the start time of the currently executing task on each machine is required (i.e., t_j). Determing the start time for a task i requires knowledge of the actual execution time of the previously executed task k on that machine to calculate task k's actual completion time. During simulations used for heuristic evaluation, our methodology utilizes the expectation of the execution time pmf as the actual execution time for each task. Thus, the start time of the subsequent task i is known, enabling the calculation of $\psi(t)$.

Taking the expectation of the class pmfs to produce the most likely actual execution times to use for the evaluation simulations is reasonable in this environment because the task execution time pmfs are assumed to be unimodal. Further study is required to determine the most effective approach when execution time pmfs are not unimodal.

A second factor in evaluating a resource allocation heuristic is the set of tasks and the ordering of task arrivals. Because of variations in the set of tasks to be executed and changes in their arrival ordering, multiple simulation trials should be conducted to adequately predict the typical relative performance among the evaluated resource allocation heuristics. In evaluating our results, we will demonstrate that in a dynamic environment a small number of simulation trials are required to sufficiently indicate the performance of a resource allocation heuristic relative to our given performance objective. Each trial involves a unique set of tasks with a unique arrival order.

To produce a dynamic SRM value for a resource allocation heuristic a relatively small number of simulations are executed where the mean of each task execution time distribution is used as the actual execution time. This produces a dynamic SRM value for each resource allocation (simulation trial) where the dynamic SRM value is determined as the average of the instaneous SRM values within that simulation trial. The dynamic SRM values for all of the simulation trials are then combined by taking their average to determine a single dynamic SRM value for the resource allocation heuristic.

The dynamic SRM values for different resource allocation heuristics can then be compared to select the approach that is more robust within the given environment. That is, given the presented formulation of the instantaneous SRM, a simulation trial that has a higher dynamic SRM value should reasonably be expected to produce fewer task deadline misses. In the next section, we present the heuristics that are to be evaluated using the dynamic SRM value in Section 7.

5. Resource Allocation Heuristics to Evaluate

5.1. Introduction

The goal of resource allocation heuristics is to select a mapping of tasks to machines and scheduling of tasks within a machine that minimizes an objective function. The presented heuristics do not directly attempt to maximize the stochastic robustness metric, instead focusing on minimizing the primary objective function. In the results section, the heuristics will be compared using both the stochastic robustness metric and their average performance relative to minimizing Equation 2. All of the heuristics were given a limited amount of time to complete a mapping event.

5.2. Two Phase Greedy

The Two Phase Greedy heuristic is based on the principles of the Min-Min algorithm (first presented in [8], and shown to perform well in many environments [7], [10], [15]). The heuristic allocates one task at each iteration, continuing until all task allocations have been resolved. In the first phase of each iteration, the Two Phase Greedy heuristic determines the best assignment (according to the performance goal) for each of the tasks left unmapped. In the second phase, it selects the task to map based on the best result found in the first phase. The completion time distribution for a given machine j at time t is denoted $\underline{F^j(t)}$. Given the set of tasks assigned to machine j at time t, denoted $\underline{MQ^j(t)}$, $F^j(t)$ can be found as follows:

$$F^j(t) = GT\{t, t_j * f_{c_1 j}\} * f_{c_2 j} * \cdots * f_{c_{|MQ^j(t)|}}. \quad (5)$$

The Two Phase Greedy heuristic is summarized in Figure 1.

while not all tasks are mapped
  for each unmapped task i
    find machine m_j such that
    $m_j \leftarrow \underset{1 \le j \le M}{\text{argmin}} \left[\mathbb{E}[F^j(t) * f_{c_i j}]\right]$;
    resolve ties arbitrarily;
  end for loop
  let $A =$ all (i, m_j) pairs found above
  select pair(s) (x, m_y) such that
  $(x, y) \leftarrow \underset{\forall (i,m_j) \in A}{\text{argmin}} \left[\mathbb{E}[F^{m_j}(t) * f_{c_i m_j}]\right]$;
  resolve ties arbitrarily;
  map x to machine y;
  update $F^y(t)$ based on assignment;
end while loop.

Figure 1. Pseudo-code describing the Two Phase Greedy heuristic.

5.3. Segmented Two Phase Greedy (STG)

The segmented two phase greedy heuristic relies on "segmenting" the collection of tasks to be allocated into n groups and then applying a two phase greedy heuristic to each group [18]. A weighting factor is used to determine segments. The new STG heuristic introduced here for this environment calculates a task's weight based on the probability of the task meeting its deadline. That is, the lower the probability that a task will complete within its deadline the higher its queuing priority will be. A weight $\underline{\pi_i}$ that is used to assign the task queueing order is defined for a task i given M machines as follows,

$$\pi_i = \frac{\sum_{j=1}^{M} \mathbb{P}[F_i(t) \le \beta_i^{max}]}{M}. \quad (6)$$

The individual weights are used to define a weighted expected time to compute for each task, denoted $\underline{\mathcal{W}_{ij}}$ for a given task i on machine j.

$$\mathcal{W}_{ij} = \pi_i \mathbb{E}[T_{c_i j}] \quad (7)$$

The weighted expected execution times are combined to produce a weighted expected completion time, denoted $\underline{\mathcal{W}_{ij}^{ECT}(t)}$ for a given task i on machine j at time t. The weighted expected completion time is calculated using Equation 8.

$$\mathcal{W}_{ij}^{ECT}(t) = \mathcal{W}_{ij} + \mathbb{E}[F^j(t)] \quad (8)$$

Tasks are sorted in ascending order according to the average of their $\mathcal{W}_{ij}$ values across all machines at the time of the mapping event. The sorted task list is then divided into n segments of equal length that are allocated to machines using a two phase greedy heuristic. The two phase greedy heuristic is used to minimize the weighted expected completion time for the last to finish task. Figure 2 presents the details of the STG heuristic discussed here.

Because a new mapping event occurs each time a task finishes, in general, each machine needs no more than two pending tasks. Thus, once every machine has two tasks pending, the heuristic can terminate the mapping event. This was utilized in the implementation of the STG heuristic to improve the heuristic's execution time.

5.4. Negotiation

Iterative approaches have been applied to static mapping problems to search the space of task permutations in a schedule [4, 5, 16]. Local search also has been applied to *dynamic* scheduling problems [12, 13]. The negotiation heuristic introduced here is modeled after such iterative

sort tasks in ascending order by average $\mathcal{W}_{ij}$ value
partition sorted task list into n segments
for each segment $\mathcal{S}$
  while not all tasks are mapped
    for each unmapped task i in $\mathcal{S}$
      find the machine m_j such that
      $m_j \leftarrow \underset{1 \leq j \leq M}{\text{argmin}} \left[\mathcal{W}_{ij}^{ECT}(t)\right]$;
      resolve ties arbitrarily;
    end for loop
    from all (i, m_j) pairs found above
    select pair(s) (x, y) such that
    $(x, y) \leftarrow \underset{\forall (i,m_j)}{\text{argmin}} \left[\mathcal{W}_{im_j}^{ECT}(t)\right]$;
    resolve ties arbitrarily;
    map task x to machine y;
    update $F^y(t)$ based on assignment;
  end while loop
end for loop.

Figure 2. Pseudo-code describing the STG heuristic.

heuristics operating in a dynamic environment. A total ordering of tasks serves as input to a schedule builder that assigns tasks to machines such that the performance objective is maximized. The total ordering is iteratively permuted and the schedule builder re-applied to produce a new resource allocation. Schedules with a higher value are kept, where value is defined by the evaluation of a fitness function. The procedure is analogous to a next-descent search in schedule space.

Each iteration of the negotiation heuristic relies on computing the following two metrics. The local earliness metric quantifies the difference between a task's expected finishing time and its deadline given the current mapping and scheduling. The local earliness metric for a given task i at time t, denoted LEM_{ij}, can be quantified as,

$$LEM_{ij} = \beta_i^{max} - \mathbb{E}[F_i(t)]. \tag{9}$$

Using the local earliness metric the global earliness metric, denoted GEM for a given mapping event, can be found as the sum of the local earliness metrics for all mappable tasks. That is, given a task ordering o and an operator $m(i)$ that returns the machine that task i has been assigned

$$GEM(o) = \sum_{\forall i \in MQ(t)} LEM_{im(i)}. \tag{10}$$

An iteration of the negotiation algorithm is defined as follows. From a current ordering ω of tasks in $MQ(t)$, two tasks are randomly selected for swapping (tasks are initially ordered by arrival times). The ordering that results from this modification, denoted ω', is used as an activity list for a schedule builder. The schedule builder assigns each task in ω', in order, to the machine that maximizes the local earliness metric as defined by Equation 9. The fitness of the resulting mapping is measured using the global earliness metric defined in Equation 10, where a higher global earliness metric indicates a more fit schedule. If the fitness of this mapping is the best encountered, the ordering ω' is kept for the next iteration. Otherwise, the ordering is discarded and the original ordering ω is maintained for the next iteration. Negotiation terminates after N iterations have been executed. The Negotiation heuristic is summarized in Figure 3.

initialize ω to an ordering of all tasks to be mapped
for N iterations
  randomly select two tasks for swapping
  swap ordering of selected tasks
  assign resultant total ordering to ω'
  execute schedule builder using ω'
  if $(GEM(\omega') < GEM(\omega))$
    $\omega \leftarrow \omega'$
end for loop.

Figure 3. Pseudo-code describing the Negotiation heuristic.

6. Simulation Setup

For this research, it is assumed that 1) the time necessary for a mapping event is negligible compared to the task arrival time, and 2) task execution times are considerably longer than the difference between successive inter-task arrival times.

To evaluate the effectiveness of the dynamic SRM value for predicting heuristic performance we considered two sets of simulations. In the first set of simulations, a small number of simulation trials were conducted to produce a dynamic SRM value for a resource allocation heuristic, as defined in our methodology. For the second set of simulations, a larger number of trials were conducted to produce sample cost values for each heuristic, where the actual execution time for each task is determined by sampling the execution time pmf for that task. For this work 10 simulation trials were used to produce a dynamic SRM value for a resource allocation heuristic as compared to 100 simulation trials to evaluate the performance metric.

All simulations consisted of 1024 tasks to be processed by eight machines, where task arrival times were not known in advance. Each arriving task belonged to one of five classes whose execution time pmfs for each machine were

known in advance. The execution time for a mapping event for all heuristics was limited to 0.1 seconds.

7. Simulation Results

The heuristics were compared using the dynamic SRM value and their ability to minimize cost, i.e., the number of tasks that miss their deadlines. Recall that the dynamic SRM value for each heuristic was constructed using a small number of independent simulation trials, where the actual execution times for tasks were set to the expectation of the task execution time pmfs.

Evaluating the success of the dynamic SRM value in comparing heuristics requires the actual performance of each heuristic over a significant number of simulation trials. Given the formulation of the dynamic SRM value, a high dynamic SRM value for a heuristic should indicate a low cost for the heuristic. In other words, heuristics that produce higher dynamic SRM values should have lower cost, i.e., have fewer task deadline misses, than those with low dynamic SRM values.

Figure 4 presents the cost distributions for each of the three heuristics. Each distribution was generated using a kernel density estimator where the kernel function was set to be Gaussian [6]. The cost results from the 100 simulation trials were used as the sample data for the kernel density estimator. Also plotted in the figure are the means of each distribution and the calculated dynamic SRM value for each heuristic. The cost distributions for the three heuristics are only valid in the interval $[0, 1024]$ corresponding to the lowest possible cost for a heuristic and the highest possible cost given the defined simulation setup.

As can be seen in Figure 4, the Negotiation heuristic produced the lowest mean cost of 87.77 and had the highest dynamic SRM value of 0.609. The Two Phase Greedy heuristic had the next lowest mean cost with 279.28 and the next highest dynamic SRM value of 0.392. Finally, the STG heuristic produced the highest mean cost of 593.6 and the lowest dynamic SRM value of 0.206.

The results of Figure 4 suggest that there is a correlation between the dynamic SRM value and the number of tasks that miss their deadlines. If there is a correlation between the dynamic SRM value and the number of deadline misses, then a plot of all of the simulation sample points taken for the three heuristics should lie on a common curve. To evaluate this conjecture, we combined the sample points for the three heuristics and applied Bayesian regression to produce Figure 5.

The sample costs taken from the three heuristics are plotted as points in the figure where triangles represent the sample points taken from the Negotiation heuristic, diamonds represent the sample points for the STG heuristic, and circles the sample points for the Two Phase Greedy heuristic. Using these data points a Bayesian regression model [6] was generated to fit the points to a common curve. The results of the regression model are plotted in Figure 5 as the the dynamic SRM value versus the logarithm of the cost. The model combines a series of radial basis functions with variance 0.2 that were uniformly distributed on the interval [0,1]. The range of cost results for the simulation trials were re-mapped to the interval [0,1] where 0 corresponds to the smallest possible cost and 1 to the highest possible cost, i.e., 1024. The line plotted in the figure represents the mean of the Bayesian model and the shaded region represents one standard deviation around the mean. The generated model appears to fit the combined sample points taken from the three heuristics to a single simple curve suggesting that there may be a correlation between changes in the dynamic SRM value and changes in cost. Next, we consider the relative performance of the individual heuristics.

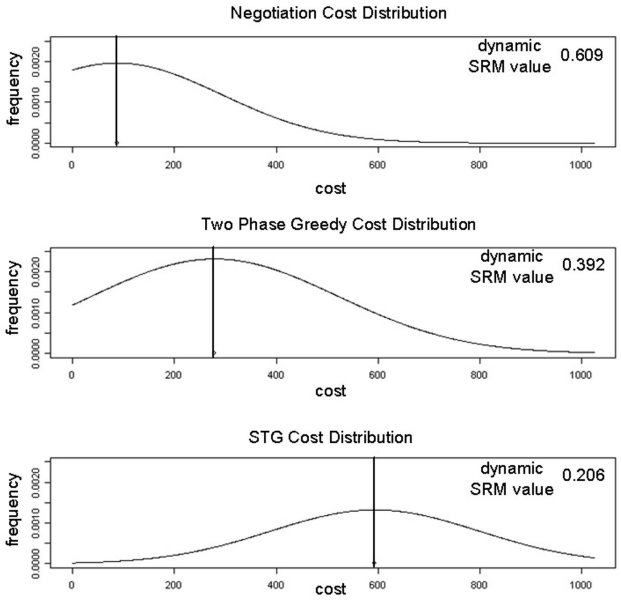

Figure 4. The distributions of cost values for all three heuristics. The cost distributions were generated using a kernel density estimator and the results of the 100 simulation trials.

From the plot of Figure 4, it appears that the negotiation heuristic generally outperforms the others. To more accurately compare the results of the three heuristics another distribution was generated to assess the frequency with which the Negotiation heuristic outperforms the STG and Two Phase Greedy heuristics. Using the resource allocation cost data taken from the 100 trials, Figure 6 shows the cost distributions with mean values for both the Negotiation

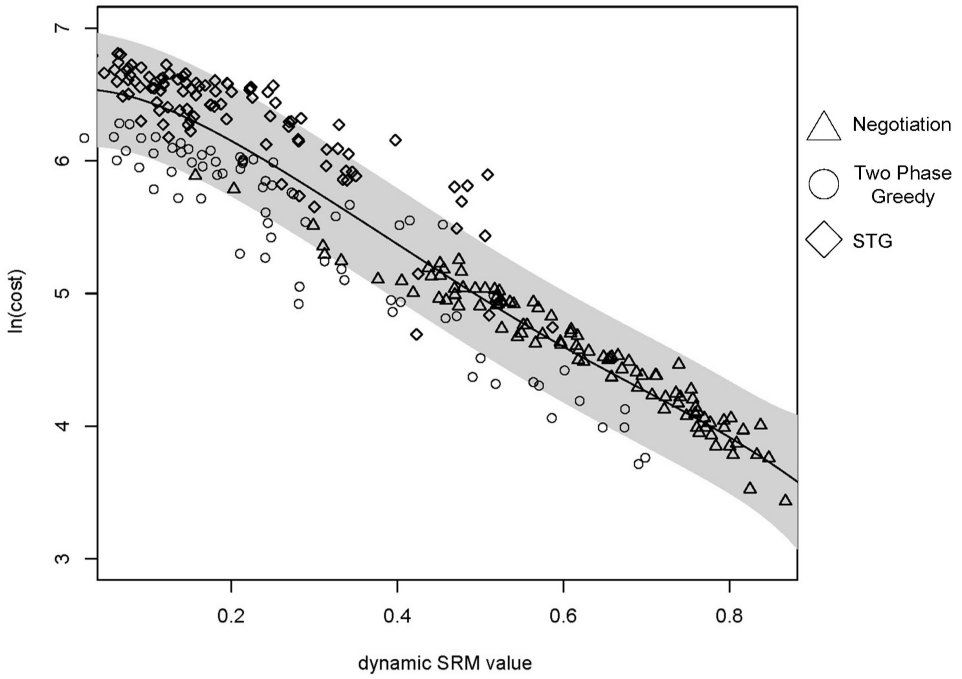

Figure 5. Plot of dynamic SRM value versus the logarithm of the costs for all three heuristics. A Bayesian regression model has been used to fit a curve to the combined set of sample points for all three heuristics. The line in the figure is the mean of the regression model and the shaded region represents one standard deviation around the mean.

heuristic and the STG heuristic and a cost comparison for the two. To generate the cost comparison distribution, labeled "STG vs. Negotiation Cost Comparison," the cost values generated for the Negotiation heuristic were subtracted from those generated by the STG heuristic for each simulation trial. The samples used to define the actual execution times were drawn in advance of the simulation trials and were the same for each heuristic. A kernel density estimator was then applied to the resultant data points to generate the distributions in the figure. For each simulation trial, the Negotiation heuristic produced a lower cost than that produced by the STG heuristic. However, for a small number of simulation trials the STG heuristic performed comparably to the Negotiation heuristic. These cost comparison values were very close to zero but still slightly positive causing the tail of the cost comparison curve generated by the kernel density estimator to edge into negative values. This suggests that there is a small probability that given the right circumstances STG might perform better than Negotiation.

Figure 7 compares the cost distribution for the Two Phase Greedy heuristic with the cost distribution for the Negotiation heuristic. As can be seen in the comparison plot, labeled "Two Phase Greedy vs. Negotiation Cost Comparison," the density estimate of the comparison cost distribution has a non-zero frequency for a sizable number of negative values. That is, for these trials the Two Phase Greedy Heuristic had fewer task deadline misses than the Negotiation heuristic. However, for the majority of the trials the Negotiation heuristic outperformed the Two Phase Greedy heuristic. From the cost comparison plot this can be seen because the mean of the comparison cost distribution is positive implying that more often than not the Two Phase Greedy heuristic had a higher cost than Negotiation.

8. Related Work

In [14], the authors define a stochastic methodology for evaluating the robustness of a resource allocation in a static environment. In that work, uncertainty in system parameters and its impact on system performance are modeled stochastically. This stochastic model was then used to derive a quantifiable measure of the robustness of a resource allocation in a static environment. This was done by defining stochastic completion times in a similar manner to our

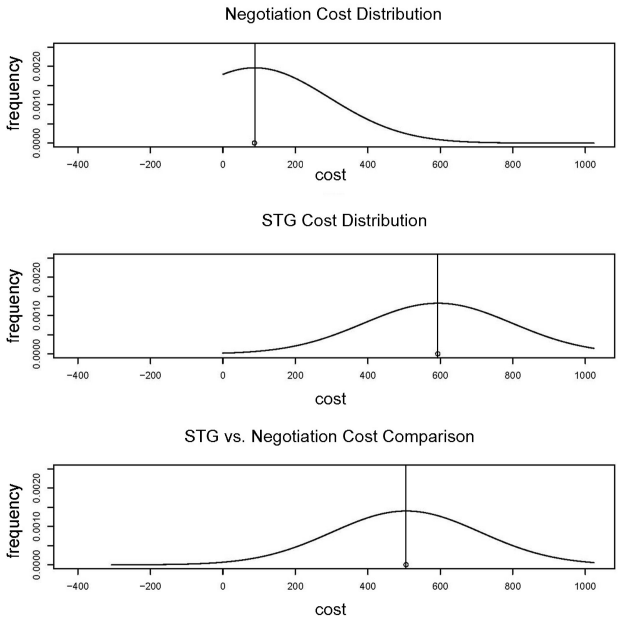

Figure 6. A comparison of the STG heuristic's cost distribution and the Negotiation heuristic's cost distribution. The comparison plot, labeled "STG vs. Negotiation Cost Comparison," shows that the Negotiation heuristic consistently performs better than the STG heuristic for all simulation trials.

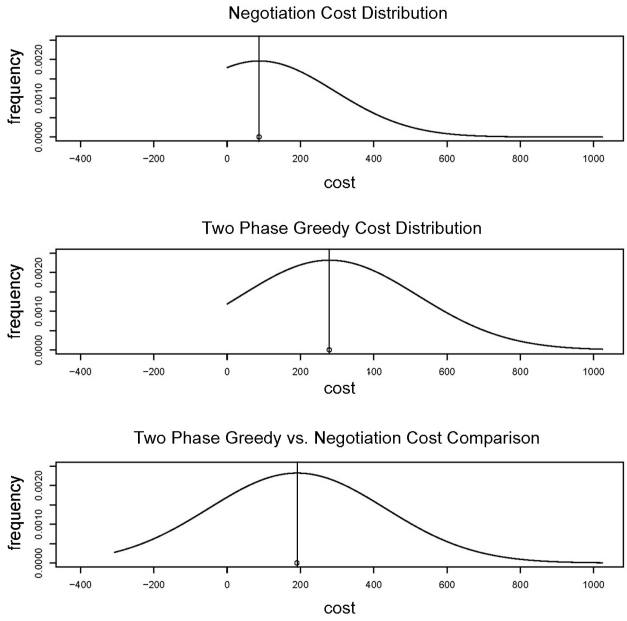

Figure 7. A comparison of the Two Phase Greedy heuristic's cost distribution and the Negotiation heuristic's cost distribution. The comparison plot, labeled "Two Phase Greedy vs. Negotiation Cost Comparison," shows that the Negotiation heuristic was not uniformly better than the Two Phase Greedy heuristic.

current presentation. A major distinction between the two formulations is that our previous work only considered a static environment where all machines are idle at the beginning of a mapping and the set of all tasks to be mapped is known in advance. In this work, the completion time calculations are very similar but machines may not be idle when a mapping event occurs and new tasks are constantly arriving. The stochastic completion time calculations are an important component of the stochastic robustness metric calculation in [14]. The result of the calculation is a probability distribution for task completion times that is then used in the calculation of the stochastic robustness metric for a resource allocation.

Intuitively, the expression of robustness presented in [14] provides a measure of the likelihood that the makespan of a resource allocation will fall within the provided bounds. This general concept has been used in this work but has been adapted to the details of the present *dynamic* environment. In this work, we were given bounds on the acceptable completion times for each task as opposed to a bound on the acceptable completion time for a collection of tasks. The derived joint probability distribution that defines the stochastic robustness metric corresponds to the probability that all tasks complete by their deadline as opposed to the probability that the collection of tasks will complete by a given deadline as in [14]. A further distinction of this work over [14] is the use of many individual measures of the resource allocation to produce a predictive measure of the robustness.

In [11], the robustness concept is used to develop resource allocations in a dynamic environment. However, that work utilizes a deterministic estimate of task execution times to develop the robustness of the resource allocation's makespan when the task execution time estimates vary from their predicted values. In this work, the robustness of a given resource allocation heuristic is instead formulated with respect to its ability to meet individual task deadlines. Another distinction between this work and [11] is that task execution times in [11] are simply deterministic execution time estimates. In contrast, this work models task execution times as random variables where we assume the existence of an empirical distribution.

9. Conclusions

Our results suggest that there is an inverse relationship between the dynamic SRM value and the performance objective of the problem studied. In reviewing our results, we explored the general relationship between the dynamic SRM value and the performance objective by analyzing the fit of a Bayesian regression model to our results. The relatively good fit of the regression model to the combined data of the three heuristics is strong evidence of the relationship between the dynamic SRM value and the stated performance objective. Our results appear to demonstrate that the dynamic SRM value can simplify the evaluation of resource allocation heuristics in a dynamic environment. That is, the methodology for determining the dynamic SRM value for a heuristic reduces the number of simulations required to demonstrate the superiority of one heuristic over another in a dynamic resource allocation environment.

Using these results we compared the performance of three different heuristics taken from the literature and applied to a stochastic dynamic environment. From this comparison, the Negotiation heuristic showed promising results in this environment. Given the apparent relationship between the dynamic SRM value and the stated performance objective, a valuable extension of this work would be the development of resource allocation heuristics that incorporate the dynamic stochastic robustness metric during a resource allocation.

References

[1] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim. Measuring the robustness of a resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):630–641, July 2004.

[2] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup web site. Technical Report HPL-1999-35R1, Hewlett Packard Corporation, CA, Sept. 1999.

[3] M. Arlitt and T. Jin. A workload characterization study of the 1998 world cup web site. *IEEE Network*, 14:30–37, May/June 2000.

[4] L. Barbulescu, A. E. Howe, L. D. Whitley, and M. Roberts. Trading places: How to schedule more in a multi-resource oversubscribed scheduling problem. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2004)*, pages 227–234, 2004.

[5] L. Barbulescu, L. D. Whitley, and A. E. Howe. Leap before you look: An effective strategy in an oversubscribed scheduling problem. In *Proceedings of the 19th National Conference on Artifial Intelligence*, pages 143–148, 2004.

[6] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA, first edition, 2006.

[7] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, June 2001.

[8] O. H. Ibarra and C. E. Kim. Heuristic algorithms for scheduling independent tasks on non-identical processors. *Journal of the ACM*, 24(2):280–289, Apr. 1977.

[9] A. Leon-Garcia. *Probability & Random Processes for Electrical Engineering*. Addison Wesley, Reading, MA, 1989.

[10] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic mapping of a class of independent tasks onto heterogeneous computing systems. *Journal of Parallel and Distributed Computing*, 59(2):107–121, Nov. 1999.

[11] A. M. Mehta, J. Smith, H. J. Siegel, A. A. Maciejewski, A. Jayaseelan, and B. Ye. Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness. *Journal of Supercomputing, Special Issue on Grid Technology*, accepted, to appear.

[12] A. J. Page and T. J. Naughton. Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium*, Apr. 2005.

[13] K. Ross and N. Bambos. Local search scheduling algorithms for maximal throughput in packet switches. In *Proceedings of the 23rd Conference of the IEEE Communications Society (INFOCOM 2004)*, 2004.

[14] V. Shestak, J. Smith, H. J. Siegel, and A. A. Maciejewski. A stochastic approach to measuring the robustness of resource allocations in distributed systems. In *2006 International Conference on Patallel Processing (ICPP 2006)*, pages 6–12, Aug. 2006.

[15] V. Shestak, J. Smith, R. Umland, J. Hale, P. Moranville, A. A. Maciejewski, and H. J. Siegel. Greedy approaches to static stochastic robust resource allocation for periodic sensor driven systems. In *Proceedings of The 2006 International Conference on Parallel and Distributed Processing Techniques and Applications*, volume 1, pages 3–9, June 2006.

[16] R. Vaessens, E. Aarts, and J. Lenstra. Job-shop scheduling by local search. Technical Report http://citeseer.ist.psu.edu/vaessens94job.html, Eindhoven University of Technology, Eindhoven, The Netherlands, 1994 (accessed May 2006).

[17] L. Wasserman. *All of Statistics: A Concise Course in Statistical Inference*. Springer Science+Business Media, New York, NY, 2005.

[18] M. Wu and W. Shu. Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems. In *Proceedings of the 9th IEEE Heterogeneous Computing Workshop*, pages 375–385, Mar. 2000.