Batch Mode Stochastic-Based Robust Dynamic Resource Allocation in a Heterogeneous Computing System

Jay Smith*†
*DigitalGlobe
Longmont, CO, USA
Email: jtsmith@digitalglobe.com

Jonathan Apodaca†, Anthony A. Maciejewski†, and H. J. Siegel†‡
†Dept. of Electrical and Computer Engineering
‡Dept. of Computer Science
Colorado State University
Fort Collins, CO, USA
Email: {jonathan.apodaca, aam, hj}@colostate.edu

***Abstract*—Heterogeneous, parallel and distributed computing systems frequently must operate in environments where uncertainty in system parameters is common. *Robustness* can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed. In such an environment, the amount of processing required to complete any given task may fluctuate substantially due to variations in data size and content. Determining a resource allocation that accounts for this uncertainty is an important area of research. In this study, we define a stochastic robustness measure to facilitate *batch*-mode resource allocation decisions in a dynamic environment where tasks are subject to individual deadlines and design a novel resource allocation technique that attempts to maximize our new stochastic robustness measure. We compare the performance of our technique against some commonly used approaches taken from the literature and adapted to our environment. Our performance results demonstrate the viability of our new technique in a dynamic heterogeneous computing system.**

Keywords: robustness, heterogeneous computing, resource management, dynamic resource allocation, distributed computing

I. Introduction

Heterogeneous, parallel and distributed computing systems frequently must operate in environments where uncertainty in system parameters is common. **Robustness** can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed [1]. In this research, we design a new stochastic robustness measure based on our earlier research [2]. Our measure is suited to facilitating batch-mode resource allocation decisions in a dynamic environment that is over-subscribed and tasks are subject to individual deadlines, i.e., the arrival rate of tasks is such that the system is not able to complete all tasks on-time.

This research was motivated by a heterogeneous distributed computing system used for processing data intensive compute tasks. In this system, user tasks are queued at a resource manager for assignment to any one of a collection of dedicated heterogeneous machines. The exact execution time of any given task is assumed dependent on the details of the data that is to be processed (including the size and actual content of the data) and the machine that is to execute the task. Task execution times may be highly variable due to their data dependence and, as such, are treated as random variables. We assume that each machine in the HC suite operates in a non-multi-tasking mode, i.e., each machine may only execute one task at a time.

Any claim of robustness for a given system must answer three fundamental questions [1]: (a) What behavior makes the system robust? (b) What are the uncertainties that the system is robust against? (c) Quantitatively, exactly how robust is the system? A robust resource allocation in this environment is one that is capable of completing all tasks by their assigned deadlines. Task execution times are a known source of uncertainty and variation in task execution times may have a significant impact on our stated performance objective. The robustness of a resource allocation can be quantified as the expected number of tasks that will complete by their deadline, as predicted at a given point in time.

From this set of requirements, we formulate a robustness measure for resource allocations using the probability that an allocation will complete a collection of tasks on-time. Our new stochastic robustness measure extends the utility of stochastic robustness to environments that are severely over-subscribed and the likelihood of having zero probability to meet any given deadline is high. We use this formulation of the stochastic robustness measure along with the alternate measure to design a batch-mode resource allocation heuristic capable of allocating a dynamically arriving set of tasks to a dedicated heterogeneous computing (**HC**) system. In general, the problem of resource allocation in the field of heterogeneous parallel and distributed computing is NP-complete (e.g., [3], [4]), therefore, the development of heuristic techniques to find near-optimal solutions represents a large body of current research (e.g., [4]–[11]).

In our prior research [2], we assumed that the resource manager operated in **immediate-mode**, i.e., tasks were assigned to machines *immediately* upon their arrival and cannot be re-allocated later. Alternatively, **batch-mode** heuristics collect arriving tasks to form a batch prior to making resource allocation decisions. A major contribution

This research was supported by the NSF under Grant ECCS-0700559 and by the Colorado State University George T. Abell Endowment.

of our current study is the design of a new robustness measure that allows for tasks that have no probability of meeting their deadline at the time they are allocated. This additional measure is crucial for the design of resource allocation techniques that are effective in severely oversubscribed environments. In addition, we design a batch-mode resource allocation technique that leverages both the stochastic robustness measure and our new robustness measure to make effective resource allocation decisions. We compare our robustness-based resource management approach to several heuristics taken from the literature and adapted to this environment. The results of our simulation study demonstrate the effectiveness of our robustness-based approach.

In the next section, we present an overview of the system model used to evaluate the chosen approach. A sampling of the related work is given in Section III. Section IV describes our mathematical model of robustness in a dynamic environment including our extension for an over-subscribed environment. The heuristic techniques adapted to this environment are presented in Section V. The details of the simulation setup used to evaluate our heuristics are listed in Section VI. Section VII provides the results of our simulation study and Section VIII concludes the paper.

II. System

The execution time of each task i, when executed alone on machine j (one of the $\boldsymbol{M}$ machines in the HC suite), is modeled as a random variable, denoted $\boldsymbol{\eta_{ij}}$. The list of tasks that the user may select from is assumed limited to a set of frequently requested algorithms such as may be found in a research lab or military environment. Consequently, we assume that the execution time random variable for each task is well characterized. That is, we assume that a probability mass function (**pmf**) is available for each task execution time random variable on each machine (determined by historical, experimental, or analytical techniques [12], [13]). In addition, each task execution time (not *completion* time) is assumed independent, i.e., there is no inter-task communication. This assumption of independence is valid for non-multitasking execution mode, which is commonly considered in the literature (e.g., [7], [14]).

In this environment, tasks arrive dynamically and the exact sequence of task arrivals is not known in advance. Each arriving task i is assigned a deadline for completion, denoted $\boldsymbol{\delta_i}$, relative to its arrival time and its average execution time. The goal of resource allocation heuristics is to maximize the number of tasks that complete by their assigned deadline. However, the system is assumed severely over-subscribed such that not all tasks can complete by their deadline. In addition, we employ a hard deadline for task completions, i.e., if the system were to fail to complete a task by its deadline, then we assume that the task has no value.

All incoming tasks are first queued at a resource manager for later assignment to any one of the M dedicated heterogeneous machines for processing. (e.g., [15]). After assignment, each task is placed in the input queue of its assigned machine and all input data for the task are staged to the machine in advance of task execution. For this study, because the data required by each task is assumed substantial, once a task has been queued for execution on a machine it cannot be reassigned and the task must be completed, even if its deadline will be missed. That is, once the system *attempts* to execute a task by assigning the task to a machine it must be completed.

Actual task execution times are dependent on the content of the data to be processed as well as its size (where the exact details of this dependence are not known in advance) and the machine that is to execute the task. We assume that an accurate pmf describing the possible execution times for each task on each of the heterogeneous machines exists and is available to the resource manager to aid in resource allocation decision making (several techniques exist for generating such a probability distribution, e.g., [13], [16]).

Uncertainty in task execution times can impact the completion times of all tasks that share the same machine for execution. For example, given multiple tasks assigned to the same machine, a longer than expected execution time for a task early in the queue may cause tasks later in the queue to miss their deadlines. This effect is obviously compounded when multiple tasks take longer than expected. To help mitigate the impacts of task execution time uncertainty, in this study, including the currently executing task, we chose to limit the number of tasks that can be queued at any single machine. Any remaining tasks are placed in a queue on the resource manager for future assignment.

By limiting the number of tasks that can be queued at each machine, at each time-step $\boldsymbol{t^{(k)}}$, we effectively define a batch of tasks at the resource manager, denoted $\boldsymbol{B^{(k)}}$, that is composed of the remaining tasks that are not already in a machine queue. Based on this definition, a mapping event occurs in the system when there is at least one task in $B^{(k)}$ and at least one machine queue that has at least one task less than the limit assigned to it.

III. Related Work

The general problem of dynamic resource allocation for independent tasks and heterogeneous computing systems was studied in [11]. The primary objective in [11] was to minimize system makespan, i.e., the total time required to complete all tasks. This objective is very different from the primary objective in our current research: maximize the number of tasks that complete by their deadlines. In addition, [11] only considered point estimates of task execution times where as our current research is focused on modeling task execution times as random variables.

In our previous research [17], similar to the environment considered in this research, each dynamically arriving task is assigned its own deadline relative to its arrival time and the execution time of each task is modelled as a random variable. However, in our previous work we focused on predicting the performance of heuristics in a stochastic dynamic environment. In this research, we are focused on utilizing a stochastic robustness measure to aid in resource allocation of a batch of tasks.

In [2], we defined a stochastic model of robustness and demonstrated its use in an immediate-mode resource allocation heuristic. In our prior work, we focused strictly on the stochastic robustness model as it pertains to an immediate-mode dynamic mapping environment. In that environment, robustness values of zero occurred relatively infrequently and did not drastically impact the performance of the immediate-mode heuristic that leveraged

robustness for decision making. However, in this research, we focus on a dynamic environment that is severely over-subscribed and design batch-mode resource allocation heuristics suitable to this environment. Because this environment is significantly over-subscribed, the frequency of zero robustness values is much greater than in our prior research, thus, forcing us to devise an alternate use of the robustness measure that accounts for zero values in a more graceful manner.

A robustness metric based on the Kolmorogov-Smirnov (**K-S**) statistic is proposed in [18]. The K-S statistic is computed for a resource allocation using the cumulative distribution function (**cdf**) over the chosen performance metric given an unperturbed system and a second cdf over the chosen performance metric given that the system has been perturbed. The authors use the magnitude of the K-S statistic to measure the deviation of a system from its expected behavior. The overall robustness of a system is then characterized by measuring the perturbed performance of the system for a number of different levels of perturbation. Consequently, the technique is well suited to the construction of a policy that accounts for variation in system performance prior to its deployment. However, in our environment, the exact mix of tasks that are to be executed is not known in advance, thus, the perturbation of the system is not easily determined in advance of system execution making the application of their robustness method problematic in a dynamic environment.

In Shi et al. [19], the authors present a resource allocation problem where the workload to be executed is described as an application consisting of a directed acyclic graph of tasks. The performance metric of interest in this system is makespan, i.e., the time required to complete all tasks on the critical path of the application. Task execution times in this system are estimated and the actual execution times may deviate considerably from their estimated values. The authors propose two robustness measures based on system slack for quantifying the robustness of a schedule: (1) relative schedule tardiness and (2) schedule miss rate. Relative schedule tardiness is calculated as the difference between the expected makespan for the schedule and the actual makespan. The schedule miss rate is based on the count of the number of schedule executions whose actual makespan is greater than the expected makespan. Neither approach to calculating robustness in [19] is based on stochastic information. In our current research, while different from the heuristics in [19], we have adapted a two phase greedy heuristic (MMU described in Section V) that employs a slack based measure to make resource allocation decisions.

IV. Background Mathematical Model

A. Stochastic Task Completion Time

In this dynamic environment, we wish to define a method for determining the completion time for a given task i at time-step $t^{(k)}$. Because the execution time of each task in the system is a random variable, the completion time of task i is found as the convolution of the execution time probability distributions [20] for all tasks either currently executing or pending execution in advance of task i on the same machine.

Let $\boldsymbol{\mu^{(k)}}$ denote the set of all tasks that are either queued for execution or currently executing on any of the M machines in the HC suite at time-step $t^{(k)}$. Let the ordered list of tasks in $\mu^{(k)}$ that were assigned to machine j in advance of request i but that have not yet completed execution as of $t^{(k)}$ be denoted $\boldsymbol{\mu_{ij}^{(k)}}$ where the tasks in $\mu_{ij}^{(k)}$ are listed in order of their assignment.

To find the completion time pmf for a currently executing task z on a given machine j, we must account for impulses in the execution time pmf of task z that would have occurred prior to the current time-step $t^{(k)}$. For example, if task z began execution at time-step $t^{(h)}$ ($h < k$), then we know that all impulses in the completion time distribution for task z with values less than $t^{(k)}$ did not occur. Thus, accurately describing the completion time of task z at time-step $t^{(k)}$ requires that these past impulses be removed from the pmf and the remaining distribution is renormalized to 1. To find the completion time distribution of task i at time-step $t^{(k)}$, we convolve the completion time distribution for the currently executing task on machine j with the execution time distributions of all pending tasks in $\mu_{ij}^{(k)}$ with the execution time distribution for task i on machine j.

B. Calculating Stochastic Robustness

To facilitate the derivation of our new robustness measure, defined in the next subsection, we summarize the details of our previous robustness calculation from [2]. The robustness of a resource allocation $\mu^{(k)}$ is defined by the joint probability that all tasks will complete by their assigned deadlines, as predicted at a given time-step $t^{(k)}$ [2]. Recall that all tasks are independent and that there is no inter-process communication among tasks. Thus, each task execution is independent, however, because each task has an individual deadline within a common machine, the joint probability of completing all tasks on-time that are assigned to the same machine is not independent. To calculate the robustness for each individual machine j, we convert the joint probability of completing all tasks by their deadline into a combination of simpler known probabilities.

Let $\boldsymbol{r_{1j}}$ denote the currently executing task on machine j at time-step $t^{(k)}$. The basis for this calculation is the known probability of completing r_{1j} by its deadline, denoted $\boldsymbol{p(r_{1j})}$. Because the completion time distribution for r_{1j} is not dependent on any of the remaining tasks assigned to this machine, we can find this probability directly from its completion time pmf as given at time-step $t^{(k)}$.

The joint probability of completing the first two tasks by their deadlines, denoted $p(r_{1j}, r_{2j})$, is given by the product of $p(r_{1j})$ and the probability that r_{2j} completes by its deadline given that r_{1j} completes by its deadline, denoted $\boldsymbol{p(r_{2j}|r_{1j})}$. To calculate $p(r_{2j}|r_{1j})$, we select the portion of the completion time distribution for r_{1j} (at time-step $t^{(k)}$) whose completion times are less than or equal to the deadline for r_{1j}. This portion of the probability distribution is then renormalized to 1 to form the pmf corresponding to $p(r_{1j})$. Convolving the distribution of $p(r_{1j})$ with the *execution* time distribution of r_{2j} gives the *completion* time distribution for r_{2j} at time-step $t^{(k)}$, given that r_{1j} completed by its deadline. The final step in determining $p(r_{2j}|r_{1j})$ is to compare the completion time distribution for r_{2j} with its deadline. That is, $p(r_{2j}|r_{1j})$

is found as the sum over the completion time distribution for r_{2j} that corresponds to r_{2j} completing by its deadline. More generally, to calculate each $p(r_{ij}|r_{i-1j})$, we extract the portion of the completion time distribution for r_{i-1j} whose completion times are less than or equal to the deadline for r_{i-1j}. This portion of the probability distribution is then renormalized to form the pmf corresponding to $p(r_{i-1j})$, denoted $\boldsymbol{D[p(r_{i-1j})]}$. Note that if $p(r_{i-1j}) = 0$, then the above calculation must terminate and return 0 because there is no probability of completing all tasks by their assigned deadline.

Given $\boldsymbol{n_j}$ application requests assigned to machine j, we iteratively apply the product rule of probability [21] as follows:

$$\begin{aligned} p(r_{1j}, r_{2j}) &= p(r_{1j})p(r_{2j}|r_{1j}) \\ p(r_{1j}, r_{2j}, r_{3j}) &= p(r_{1j})p(r_{2j}|r_{1j})p(r_{3j}|r_{1j}, r_{2j}) \\ \vdots &= \vdots \\ p(r_{1j}, r_{2j}, \cdots, r_{n_jj}) &= p(r_{1j})p(r_{2j}|r_{1j}) \cdots \\ & \quad p(r_{n_jj}|r_{1j}, r_{2j}, \cdots, r_{n_{j-1}j}). \end{aligned}$$

Because of the independence across machines in the heterogeneous suite, we can define the **stochastic robustness** of a resource allocation at a given time-step $t^{(k)}$, denoted $\boldsymbol{\rho^{(k)}}$, as the product of the joint probability associated with each machine, i.e., the probability of all machines completing their tasks by their deadlines is the product of each machine finishing its tasks by their deadlines. Let $\boldsymbol{\rho_j^{(k)}}$ denote the robustness of machine j at time-step $t^{(k)}$, i.e., $\rho_j^{(k)} = p(r_{1j}, r_{2j}, \cdots, r_{n_jj})$ Formally,

$$\rho^{(k)} = \prod_{\forall j} \rho_j^{(k)}. \tag{1}$$

C. Expected Number of On-time Completions

In the above formulation of robustness, given a resource allocation $\mu^{(k)}$, if any task in $\mu^{(k)}$ were to miss its deadline prior to its completion, then the robustness of $\mu^{(k)}$ would be 0. However, in our resource allocation problem, our goal is to maximize the number of tasks that complete before their deadline and the system is considered severely over-subscribed. Consequently, a resource allocation where, for example, only 1 task misses its deadline is more valuable than a resource allocation where 10 tasks miss their deadline. However, using only the above formulation of stochastic robustness, both allocations would have the same robustness value of 0 regardless of the number of tasks that each completes on-time.

To extend stochastic robustness into environments that are over-subscribed, we can account for tasks where $p(r_{ij}) = 0$ separately by removing them from the robustness calculation. Intuitively, when we remove a single task from the stochastic robustness calculation, we are approximating the probability that all but one task will complete by their deadline instead of all of the tasks. Using this technique, we can create an alternate measure of robustness that computes the expected number of tasks to make their deadline for a given resource allocation. In this quantification of robustness, if the probability of completing a given task i on machine j is 0, then we do not include the task in our joint probability calculation as before. Instead, we account for the missed task deadline separately and proceed with our robustness calculation as in the previous subsection where we use the completion time distribution of task r_{ij}, denoted $\boldsymbol{C(r_{ij})}$, in place of the probability distribution for $p(r_{ij})$ (which would be 0). Let $\boldsymbol{m_j}$ denote the number of tasks that miss their deadline on machine j at time-step $t^{(k)}$. Thus, in this quantification of robustness, we limit the joint probability calculation of $\rho_j^{(k)}$ to the $n_j - m_j$ tasks that have a non-zero probability of completing by their deadline.

The pseudo-code describing this procedure is listed in Algorithm 1. Given n_j tasks pending or executing on machine j at time-step $t^{(k)}$, the expected number of task completions can be found in the following manner. As before, let r_{1j} denote the currently executing task on machine j, let $\boldsymbol{C_j}$ denote the completion time of task i on machine j, and let **robust**$\boldsymbol{(\delta_x, x)}$ denote the procedure for finding $D[p(x)]$.

We begin by setting $\rho_j^{(k)}$ to 1 and m_j to 0. We first find $C(r_{1j})$ as before and calculate $p(r_{1j})$. If $p(r_{1j}) = 0$, then we increment m_j by one and proceed using $C(r_{1j})$ in place of $D[p(r_{1j})]$ leaving $\rho_j^{(k)}$ unmodified. If $p(r_{1j}) > 0$, then we proceed to the next step using the pmf for $p(r_{1j})$ and multiply ρ_j by $p(r_{1j})$. In general, for a given task r_{ij}, if $p(r_{ij}) > 0$, then we proceed to the next task using the pmf for $p(r_{ij})$ and multiply $\rho_j^{(k)}$ by $p(r_{ij})$. Alternatively, if $p(r_{ij}) = 0$, then we proceed using $C(r_{ij})$, leave $\rho_j^{(k)}$ unmodified, and increment m_j by one. The procedure continues through task r_{n_jj}. Using m_j, $\rho_j^{(k)}$, and n_j, we compute the expected number of tasks to complete by their deadline as $\rho_j^{(k)}(n_j - m_j)$.

V. Heuristics

The total number of expected on-time task completions at time-step $t^{(k)}$ is given as

$$\sum_{\forall j} \rho_j^{(k)}(n_j - m_j). \tag{2}$$

Thus, the overall robustness of a resource allocation $\mu^{(k)}$ can be quantified as the count of the tasks in $\mu^{(k)}$ that are expected to complete on-time.

A. MinCompletion-MinCompletion (MM)

The MinCompletion-MinCompletion (**MM**) heuristic was adapted from Algorithm E in [4]. The heuristic operates in two basic phases where in phase 1 the heuristic identifies the machine that maximizes the performance objective for each task ignoring the others. In phase 2, the heuristic identifies the task-machine pairing that maximizes the performance objective over all task machine pairs identified in phase 1.

In the MM heuristic, the performance objective of each phase is to minimize the expected completion time. At each mapping event $t^{(k)}$, MM first copies the batch to a separate queue $\boldsymbol{Q}$ and finds the machine j that provides the minimum expected completion time for each task in Q. From this set of task-machine pairs, MM selects the pair that provides the overall minimum expected completion time and provisionally assigns the task to its selected

Algorithm 1 Pseudo-code for calculating the number of tasks that are expected to complete on time for a given machine j. The symbol $*$ is meant to denote convolution.

$\rho_j^{(k)} \leftarrow 1$
$m \leftarrow 0$
$i = 1$
$D[p(r_{1j})] \leftarrow \text{robust}\,(\delta_1, C(r_{1j}))$
if $p(r_{1j}) > 0$ **then**
  $C_j \leftarrow D[p(r_{1j})]$
else
  $C_j \leftarrow C(r_{1j})$
end if
while $i < n_j$ **do**
  $i \leftarrow i + 1$
  $D[p(r_{ij})] \leftarrow \text{robust}\,(\delta_i, C_j * r_{ij})$
  if $p(r_{ij}) = 0$ **then**
    $m_j \leftarrow m_j + 1$
    $C_j \leftarrow C_j * r_{ij}$
  else
    $\rho_j^{(k)} \leftarrow \rho_j^{(k)} \times p(r_{ij})$
    $C_j \leftarrow D[p(r_{ij})]$
  end if
end while
return $\rho_j^{(k)} \times (n_j - m_j)$

machine. The process is repeated until all of the tasks in Q have been provisionally assigned. To complete the mapping event, MM iterates through all machine queues and for each queue where the current length is less than four, tasks are moved from the provisional assignment to the available machine queue until the length of the machine queue is equal to the limit. The stopping criteria for the procedure occurs when there are no tasks left in Q, there are no remaining machines with free slots, or there are no tasks that get assigned to a machine with a free slot.

The basic pseudo-code for the two-phase greedy heuristic is listed in Algorithm 2. In the MM heuristic, the phase1 procedure returns the machine that minimizes the expected completion time for the provided task. The phase2 procedure returns the task-machine pair that provides the overall minimum expected completion time for all pairs identified in phase1.

B. MinCompletion-Maximum Urgency (MMU)

The MinCompletion-Maximum Urgency (**MMU**) heuristic is also a two phase greedy heuristic that operates using expected execution times and limits the number of tasks pending completion on each machine to four. We define task urgency as 1 over the difference between the expected completion time for the task and its deadline. Effectively, MMU emphasizes allocating tasks with the minimum slack. That is, given task i assigned to machine j, the urgency of task i is given as

$$\frac{1}{\delta_i - \mathbb{E}[C(r_{ij})]}. \tag{3}$$

Algorithm 2 Pseudo-code for the two phase greedy procedure.

1: $Q \leftarrow B^{(k)}$
2: **while** stopping criteria not met **do**
3: pairs $\leftarrow \emptyset$
4: **for** task t_i in Q **do**
5: $m_j \leftarrow$ phase1(t_i)
6: pairs $\leftarrow$ pairs $\cup\, (t_i, m_j)$
7: **end for**
8: $(t_x, m_y) \leftarrow$ phase2(pairs)
9: **if** queue size for $m_y < 4$ **then**
10: map t_x to m_y
11: **end if**
12: remove t_x from Q
13: **end while**

In the first phase of MMU, the heuristic identifies the minimum expected completion time machine for each task in $B^{(k)}$. In the second phase, based on the task completion times found in phase 1, MMU selects the assignment whose task urgency is the greatest, i.e., has the smallest slack. The pseudo-code for the MMU heuristic is given by Algorithm 2 where phase1 is defined as the minimum expected completion time and phase2 is defined as the maximum expected urgency.

C. MinCompletion-SoonestDeadline (MSD)

The Limited Queue Expected MinCompletion-Soonest Deadline heuristic (MSD) is a variation of the two phase greedy heuristic where we favor tasks with the soonest deadline. In the first phase, the heuristic selects the machine that provides the minimum expected completion time for each task ignoring the others. In the second phase, from the list of potential task-machine pairs found in the first phase, the heuristic makes the provisional assignment whose task has the soonest deadline. The pseudo-code for the MSD heuristic is given by Algorithm 2 where phase1 is defined as the expected minimum completion time procedure and phase2 is defined as the soonest deadline. In the event that two tasks have the same deadline and require the same machine, ties are broken by assigning the task that has the minimum expected completion time.

D. Maximum On-time Completions (MOC)

The Maximum On-time Completion heuristic (**MOC**) also operates in two phases. However, in the first phase, for each task, we identify the machine that will provide the task with the highest probability of completing by its deadline (ρ_j from Algorithm 1). From the set of task-machine pairs identified in phase 1, MOC first sorts the collection of assignments based on the machine required for the assignment. For each machine j where the number of tasks pending completion is less than four, MOC identifies the three tasks (from the task-machine pairs identified in step 1) that have the highest probability of an on-time completion that is greater than 30%. Using this list of tasks, MOC iterates through all possible orderings of these tasks on machine j to identify the ordering that has the highest number of expected on-time completions using the value returned by Algorithm 1. Using the identified

ordering, MOC assigns the first task in the ordering to its selected machine and returns the remainder of the tasks to $B^{(k)}$. The procedure continues until either there are no tasks left in $B^{(k)}$, there are no remaining machines with free slots, or there are no tasks that get assigned to a machine with a free slot.

VI. Simulation Setup

Our simulation environment consisted of eight machines (i.e., $M = 8$) that collectively exhibited inconsistent heterogeneous performance [22]; e.g., machine A may be better than machine B for application 1 but not for application 2. In our simulation study, the task execution time distributions are assumed to be unimodal. The distributions were generated based on the gamma distribution where the mean of the gamma distribution was set based on execution time results for the 12 SPECint benchmark applications for a sample set of eight machines. Using these distributions, we generated 500 random sample execution times for each application on each machine [23] where the scale parameter of each gamma distribution was selected uniformly at random from the range [1,20]. After generating the sample execution times, we applied a histogram [13] to the result to produce probability mass functions that approximate the original probability density functions—one for each application on each machine. Each benchmark application served as a model for each task type to be executed by the system, creating an eight machine by twelve task type matrix of execution time pmfs.

Our simulation study consisted of 20 independent trials. In each simulation trial, task arrivals were assumed to follow a Poisson process where the system as a whole was over-subscribed, i.e., the system is unable to complete all tasks by their deadlines. The exact mix of tasks to be executed was selected uniformly at random from the available task types. The actual task execution times were found by sampling the appropriate pmf describing the possible execution times for the task type and machine combination. In each simulation trial, an absolute deadline for each request was established as the sum of the arrival time of the request and the average expected execution time across the four best machines, i.e., the four machines with the shortest average task execution times.

Each simulation trial included 2000 tasks that arrived over a period of approximately 20,000 time-steps. In all trials, the arrival rate of tasks was such that the system remained over-subscribed and none of the heuristics tried were able to complete all tasks by their deadline.

VII. Results

Simulation results for the heuristics are presented in Figure 1. From the figure, we can see that the MOC heuristic significantly outperforms the other heuristics, completing an average of 1401 tasks on-time for each trial (out of a possible 2000) with a 95% confidence interval of ± 19 tasks.

For comparison, we also provide the results for an immediate-mode heuristic based on minimum expected completion times. In MECT, each task is mapped immediately upon its arrival to the machine that provides the minimum expected completion time for the task. As can be seen from the plot of our results, MECT is unable to complete a significant portion of the tasks on-time, suggesting the difficulty of the resource allocation problem. On average MECT completed only 100 tasks on-time out of a possible 2000 with a 95% confidence interval of ± 5 tasks.

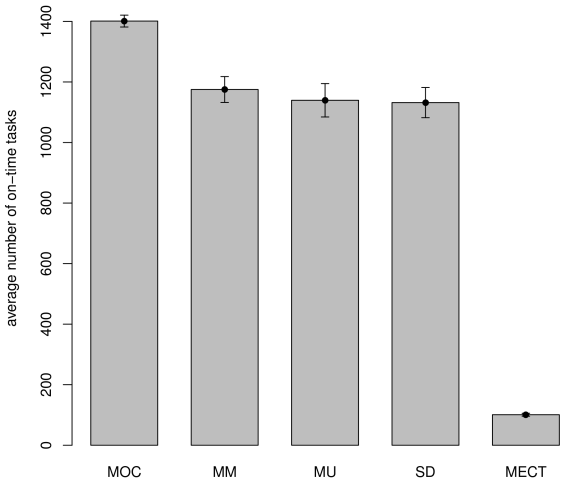

Fig. 1. A comparison of our heuristic results over 20 simulation trials where task execution time distributions are based on unimodal distributions. The results achieved for each heuristic are plotted along with their 95% confidence intervals.

The remaining heuristics all performed comparably with MM performing the best on average, completing an average of 1175 tasks by their deadline with a 95% confidence interval of plus or minus 42 tasks. The MMU heuristic performed comparably by completing an average of 1139 tasks on-time with a 95% confidence interval of ± 55 tasks. The MESD heuristic performed surprisingly well completing an average of 1131 tasks on-time with a 95% confidence interval of ± 49 tasks.

The relatively good performance of these heuristics was surprising. The limit on the number of preassigned tasks may have enabled them to reduce the amount of uncertainty in each of the completion time calculations, thus, allowing them to complete a significant number of tasks on time. Further, by focusing on first completing those tasks whose completion times are the smallest, MM naturally tends to favor tasks that can be completed by their deadline and that have the smallest impact on the remaining tasks. However, this combination limits the performance of MM in an over-subscribed environment such as ours. For example, given two tasks that are competing for the same machine where one has a larger completion time than the other, MM will always assign the smaller task first. However, throughout the simulation trials there are a significant number of situations where assigning the larger task first would result in MM completing both tasks on time instead of just one.

Comparing the results of the MM heuristic with the immediate-mode MECT heuristic suggests the over all value of our batch-mode approach for this environment.

That is, both MM and MECT are based on expected execution time values, however, MM performs dramatically better than MECT. By extending the two-phase greedy approach with our robustness models, we are able to further improve our results by an additional 19%.

VIII. Conclusions

In this work, we designed a model of stochastic robustness that facilitates its calculation and use during resource allocation in an over-subscribed dynamic resource allocation environment. We applied this model of stochastic robustness to the design of a novel resource allocation heuristic capable of assigning requests to machines in a manner that minimizes the number of requests that miss their deadline. The MOC heuristic showed significant promise for achieving this desired result in an over-subscribed dynamic environment.

Our results demonstrate the advantages of a robustness-based resource allocation approach in a stochastic environment. This research also demonstrates the viability of creating new resource allocation heuristics based on stochastic robustness in a dynamic environment. Extensions to this work may consider the impacts of misleading pmfs on heuristics that attempt to leverage the stochastic robustness model to make resource allocation decisions. In addition, we wish to further explore the use of queue limits to mitigate the impact of uncertainty in task execution times.

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*, vol. 15, no. 7, pp. 630–641, Jul. 2004.

[2] J. Smith, E. K. P. Chong, A. A. Maciejewski, and H. J. Siegel, "Stochastic-based robust dynamic resource allocation in a heterogeneous computing system," in *Proceedings of the 38th International Conference on Parallel Processing (ICPP-2009)*, Sep. 2009.

[3] E. G. Coffman, Ed., *Computer and Job-Shop Scheduling Theory*. New York, NY: John Wiley & Sons, 1976.

[4] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, Apr. 1977.

[5] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, *Characterizing Resource Allocation Heuristics for Heterogeneous Computing Systems*, ser. Advances in Computers. Amsterdam, The Netherlands: Elsevier, 2005, pp. 91–128.

[6] A. Burns, S. Punnekkat, L. Strigini, and D. R. Wright, "Probabilistic scheduling guarantees for fault-tolerant real-time systems," in *Proceedings of DCCS-7,IFIP International Working Conference on Dependable Computing for Critical Applications*, 1999, pp. 361–378.

[7] A. Dogan and F. Özguner, "Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems," *Cluster Computing*, vol. 7, no. 2, pp. 177–190, Apr. 2004.

[8] M. M. Eshaghian, Ed., *Heterogeneous Computing*. Norwood, MA: Artech House, 1996.

[9] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427–1436, Nov. 1989.

[10] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," in *Proceedings of the 4th IEEE Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 30–34.

[11] 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*, vol. 59, no. 2, pp. 107–121, Nov. 1999.

[12] Y. A. Li, J. K. Antonio, H. J. Siegel, M. Tan, and D. W. Watson, "Determining the execution time distribution for a data parallel program in a heterogeneous computing environment," *Journal of Parallel and Distributed Computing*, vol. 44, no. 1, pp. 35–52, Jul. 1997.

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

[14] A. Kumar and R. Shorey, "Performance analysis and scheduling of stochastic fork-join jobs in a multicomputer system," *Transactions on Parallel and Distributed Systems*, vol. 4, no. 10, Oct. 1993.

[15] E. Elmroth and J. Tordsson, "An interoperable, standards-based grid resource broker and job submission service," in *Proceedings of the First International Conference on e-Science and Grid Computing (e-science '05)*, 2005.

[16] M. A. Iverson, F. Özguner, and L. Potter, "Statistical prediction of task execution times through analytical benchmarking for scheduling in a heterogeneous environment," *IEEE Transactions on Computers*, vol. 48, no. 12, pp. 1374–1379, Dec. 1999.

[17] J. Smith, L. D. Briceño, A. A. Maciejewski, and H. J. Siegel, "Measuring the robustness of resource allocations in a stochastic dynamic environment," in *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS 2007)*, Mar. 2007.

[18] D. England, J. Weissman, and J. Sadagopan, "A new metric for robustness with application to job scheduling," in *Proceedings of the 14th IEEE High Performance Distributed Computing (HPDC 05)*, Jul. 2005, pp. 135–143.

[19] Z. Shi, E. Jeannot, and J. Dongarra, "Robust task scheduling in non-deterministic heterogeneous computing systems," in *Proceedings of the 2006 IEEE International Conference on Cluster Computing*, Sep. 2006, pp. 135–143.

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

[21] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed., B. S. M. Jordan, J. Kleinberg, Ed. 233 Spring Street, New York, NY 10013, USA: Springer Science+Business Media, LLC, 2006.

[22] 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*, vol. 61, no. 6, pp. 810–837, Jun. 2001.

[23] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering, Special 50th Anniversary Issue*, vol. 3, no. 3, pp. 195–207, Nov. 2000.