2009 International Conference on Parallel Processing

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

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

Edwin K. P. Chong†‡, Anthony A. Maciejewski†, and H. J. Siegel‡§
†Dept. of Electrical and Computer Engineering
‡Dept. of Mathematics
§Dept. of Computer Science
Colorado State University
Fort Collins, CO, USA
Email: {echong, aam, hj}@colostate.edu

***Abstract*—This research investigates the problem of robust dynamic resource allocation for heterogeneous distributed computing systems operating under imposed constraints. Often, such systems are expected to function in an environment where uncertainty in system parameters is common. In such an environment, the amount of processing required to complete an application may fluctuate substantially. Determining a resource allocation that accounts for this uncertainty—in a way that can provide a probability that a given level of service is achieved—is an important area of research. We define a mathematical model of stochastic robustness appropriate for a dynamic environment that can be used during resource allocation to aid heuristic decision making. In addition, we design a novel technique for maximizing stochastic robustness in this environment. Our performance results for this technique are compared with several well known resource allocation techniques in a simulated environment that models a heterogeneous distributed computing system.**

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

## I. Introduction

This work was motivated by a heterogeneous parallel and distributed computing system used for image processing. In this system, user requests for processing are queued to a resource manager for assignment to any one of a collection of dedicated machines. Each request consists of an application to be executed (e.g., compression, decompression, rotation) and an image to be processed. The list of available image processing applications that the user may select from is limited to a set of frequently requested algorithms such as may be found in a research lab or military environment.

Often, heterogeneous, distributed computing systems must operate in environments where uncertainty in system parameters is common. 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]. In this research, we define a stochastic robustness metric [2], [3] for quantifying the robustness of a resource allocation in a stochastic *dynamic* environment. Intuitively, the stochastic robustness metric gives the probability that a given resource allocation will complete all assigned applications by their deadline.

Any claim of robustness for a given system must answer these three fundamental questions [4]: (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? Robust behavior for a resource allocation in this environment can be expressed in terms of the number of applications that complete by their assigned deadline. That is, a robust resource allocation is one that is capable of completing all applications by their assigned deadline. Application execution times are a known source of uncertainty in the system and may have an impact on our stated performance objective. The robustness of a resource allocation can be quantified as the joint probability that all applications will complete by their deadline, as predicted at a given point in time.

In this environment, the exact execution time of any given application is dependent on the details of the data (including the size and actual content) that is to be processed by the application and the machine that is to execute the application. Thus, the execution times of these applications may be highly variable and, as such, are treated as random variables. Because the list of applications that may be requested is limited, the execution time random variable for each application is assumed to be well characterized. That is, we assume that a probability mass function (pmf) is available for each application execution time random variable on each machine (determined by historical, experimental, or analytical techniques [5], [6]).

In this system, users submit requests to process a provided data set by one of a set of well-known applications. The exact sequence of user requests for processing are unknown prior to their arrival, i.e., request arrival times are not known in advance. Each arriving

This research was supported by the NSF under Grants CNS-0615170 and ECCS-0700559 and by the Colorado State University George T. Abell Endowment.

0190-3918/09 $26.00 © 2009 IEEE
DOI 10.1109/ICPP.2009.45

request is assigned a deadline based on an agreement between the service provider and the customer. The time of each deadline is determined relative to the arrival time of the request, the size of the data that is to be processed, and the expected execution time of the application requested (averaged across all machines). The service provider requires that each request complete by its assigned deadline. However, if the system were to fail to complete a request by its deadline, then the request must be completed on a "best effort" basis, i.e., as close to the original deadline as possible. That is, the first commitment of the service provider is to complete the request and second to complete the request by an agreed to deadline, i.e., an instance of a soft deadline [7].

From this set of requirements, we formulate a robustness metric for resource allocations in terms of the probability that the allocation will complete all assigned requests by their deadlines. We use this formulation of the stochastic robustness metric to design a resource allocation heuristic capable of allocating a dynamically arriving set of application requests to a 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., [8], [9]), therefore, the development of heuristic techniques to find near-optimal solutions represents a large body of current research (e.g., [9]–[17]).

The major contributions of this work include: (1) a mathematical model for quantifying the robustness of resource allocations in a stochastic dynamic environment that can be used *during* resource allocation to guide the resource allocation and (2) the design of a novel resource allocation technique based on this formulation of robustness. Specifically, we propose a novel technique that attempts to greedily maximize the robustness of the system, referred to as the MaxRobust heuristic. Our results in Section VI clearly suggest the viability of this approach in this environment through comparison with a number of commonly used techniques.

In the next section, we present an overview of the system model used to evaluate the chosen approach. Section III describes our mathematical model of robustness in a dynamic environment. The MaxRobust heuristic based on this model of robustness is presented in Section IV along with four other commonly used techniques. The details of the simulation setup used to evaluate our heuristics are listed in Section V. Section VI provides the results of our simulation study. A sampling of the related work is given in Section VII and Section VIII concludes the paper.

## II. System

The computing system used in this work is comprised of a collection of heterogeneous machines that are dedicated to executing a dynamically arriving collection of image processing application requests. Incoming application requests are queued at a resource manager for assignment to a machine for processing (e.g., [18]). Each request has three elements: the application to be executed, the data that is to be processed by that application, and a deadline for completing the processing. After assignment, an application request is placed in the input queue of its assigned machine and any required data are staged to the machine in advance of application execution. We assume that once a request has been assigned to a machine for processing it cannot be reassigned to another machine. We also make the simplifying assumption that any data required to complete a request *can* be pre-staged before the machine begins processing the associated request.

The set of applications to be executed is assumed to be limited to a collection of frequently run applications, such as may be found in a military or research lab environment. The actual execution time of each application is dependent on the content and size of the data that is to be processed (where the exact details of this dependence are not known in advance) and the machine that will execute the application. However, we assume that an accurate pmf describing the possible execution times for each application on each of the heterogeneous machines exists and is available to the resource manager to aid in resource allocation (several techniques exist for generating such a probability distribution, e.g., [6], [19]).

The system is required to complete each request for processing by its assigned deadline. If the system fails to complete a request by its deadline, then the system is penalized a fixed amount for each failed request. Recall that the system is required to complete all application requests. For requests that fail to complete by their deadline, the system is expected to make a "best effort" to complete the request as close to the deadline as possible. Assuming that requests will be completed on a best effort basis implies that request assignments will not be modified after it is known that there is zero probability of completing the request by its deadline because the deadline has already passed. The goal of resource allocation heuristics in this environment is to minimize the number of requests that miss their deadline.

## III. Mathematical Model

### A. Stochastic Application Completion Time

We assume that the execution time of each request can be characterized by the application that has been requested. The execution time of each application $i$, when executed alone on machine $j$ (one of the $M$ machines in the HC suite), is modeled as a random variable, denoted $\eta_{ij}$. In addition, each application execution time (not *completion* time) is assumed independent, i.e., there is no inter-application communication. This assumption of independence is valid for non-multitasking execution mode, which is commonly considered in the literature (e.g., [12], [20]), and applied in practice in a variety of systems, e.g., an iterative universal datagram protocol server model [21].

We assume that the probability distribution describing the random variable $\eta_{ij}$ has been created from measurements of the response times of actual application executions. A typical method for creating such distributions relies on a histogram estimator [6] that produces a discrete probability distribution known as a probability mass function. Each application is associated with a set of pmfs, one pmf for each machine in the HC suite, describing the probability of all possible execution times for that application. We assume that the collection of application execution time pmfs have been provided in advance and that all of the applications that the system may be asked to execute are known *a priori*. Finally,

each request $i$ is assigned an absolute deadline for completion, denoted $\delta_i$.

At time-step $t^{(k)}$, we want to predict when machine $j$ will complete all of the requests that it has been assigned up to that point. If request $i$ is the last entry in the input queue of machine $j$ this corresponds to the completion time for request $i$. This requires a means of combining the execution times for all requests assigned to that machine. Using a deterministic (i.e., non-probabilistic) model of task execution times, the estimated execution times for all requests assigned to machine $j$ would be summed with the machine ready time to produce a completion time. A similar procedure is followed using a stochastic model as well. However, calculating stochastic completion times requires the summation of random variables as opposed to deterministic estimated values. A summation of random variables can be found as the convolution of their corresponding pmfs [22], [23].

Let $MQ(t^{(k)})$ be the set of all requests that are either queued for execution or are currently executing on any of the $M$ machines in the HC suite at time-step $t^{(k)}$. To determine the completion time for request $i$ on machine $j$ at time-step $t^{(k)}$, identify the subset of requests in $MQ(t^{(k)})$ that were assigned to machine $j$ in advance of request $i$ that have not yet completed execution, denoted $MQ_{ij}(t^{(k)})$. The execution time pmfs for the requests in $MQ_{ij}(t^{(k)})$ will be convolved with the execution time distribution for request $i$ on machine $j$ to produce the stochastic completion time pmf for request $i$ on machine $j$.

The execution time pmf for the currently executing request $z$ on machine $j$ requires some additional processing prior to its convolution with the pmfs of the queued requests to create a completion time pmf. For example, if $z$ began execution at time-step $t^{(h)}$ ($h < k$), some of the impulse values of the pmf describing the completion time of $z$ may be in the past. Therefore, to accurately describe the completion time of request $z$ at time $t^{(k)}$ 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 $z$ on machine $j$ as predicted at time-step $t^{(k)}$. To simplify notation, define an operator $GT(s, d)$ 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 request on machine $j$ is determined by applying the $GT$ operator to its completion time pmf, using the current time-step $t^{(k)} = s$. The resulting distribution is then convolved with the machine $j$ execution time pmfs in $MQ_{ij}(t^{(k)})$ and the pmf of request $i$ to produce the predicted completion time pmf for request $i$ on machine $j$ at the current time-step $t^{(k)}$.

### B. Calculating Robustness

The robustness of a resource allocation in this environment is defined by the joint probability that all applications will complete by their assigned deadline as predicted at a given time-step $t^{(k)}$. To calculate the robustness for each machine $j$, we calculate the joint probability of completing all applications assigned to this machine by iteratively applying the product rule of probability [24] to convert the joint probability of completing all applications by their deadline into a combination of simpler known probabilities. Let $r_{1j}$ denote the currently executing request 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 $p(r_{1j})$. Because the start time of the currently executing request is known and its completion time distribution is not dependent on any of the remaining requests assigned to this machine, we can find this probability directly from its completion time pmf.

The joint probability of completing the first two requests by their deadlines, denoted $p(r_{1j}, r_{2j})$, can be expressed as 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 $p(r_{2j}|r_{1j})$. To calculate $p(r_{2j}|r_{1j})$, we select the portion of the completion time distribution for $r_{1j}$ whose completion times are less than or equal to the deadline for $r_{1j}$. This portion of the probability distribution is then renormalized 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 $r_{2j}$, 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 $p(r_{i-1j})$.

Given $n_j$ application requests assigned to machine $j$, we iteratively apply the product rule of probability as follows:

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

At any time-step $t^{(k)}$, each request in the resource allocation has been previously assigned to the input queue of some machine $j$ ($1 \leq j \leq M$). Recall that there are no explicit inter-application dependencies—the only dependence that may exist between applications is their reliance on the same machine for execution. 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 $\rho^{(k)}$, as the product of the joint probability associated with each machine, i.e., the probability of all machines completing their application requests by their deadline is the product of each machine finishing its application

requests by their deadline. Formally,

$$\rho^{(k)} = \prod_{\forall j} p(r_{1j}, r_{2j}, \cdots, r_{n_j j}). \quad (1)$$

## IV. Heuristics

### A. Overview

We investigate a number of heuristics to understand the complexity required to generate robust resource management techniques in this environment. Three of the heuristics described below (MECT, MEET, and KPB) have been adapted from heuristics taken from the literature and provide points of comparison with the MaxRobust heuristic. These comparison heuristics have generated promising results in other environments and can be expected to perform reasonably well in this environment.

### B. MaxRobust

The MaxRobust heuristic applies a simple greedy approach to maximizing the stochastic robustness of the resource allocation. Upon the arrival of a new request $r_i$ at time-step $t^{(k)}$, MaxRobust assigns request $i$ to the machine that maximizes $\rho^{(k)}$ at time-step $t^{(k)}$. That is, MaxRobust calculates the $\rho^{(k)}$ value of request $i$ as it were assigned to the end of the input queue of machine $j$, for each $j = 1, \ldots, M$, and selects the machine that maximizes $\rho^{(k)}$. In this way, MaxRobust greedily assigns applications to maximize the joint probability that all applications complete by their deadline at time-step $t^{(k)}$. If all of the machines provide an identical robustness value, then MaxRobust will break the tie using the KPB heuristic presented below.

### C. Minimum Expected Completion Time (MECT)

The Minimum Expected Completion Time (MECT) heuristic (based on the Minimum Completion Time heuristic, e.g., presented in [16], [25], [26]) ignores the robustness of each allocation. Instead it allocates requests such that the expected completion time of its associated application is minimized. Using the expected execution time for each application, the expected completion time for any request $i$ on machine $j$ can be found by summing the expected execution time of each request in $MQ_{ij}(t^{(k)})$. If the input queue of machine $j$ is empty at the time of evaluation, then the expected execution time of request $i$ is summed with the current time to produce a completion time. Using the expected value of the completion time for request $i$ on each machine $j = 1, \ldots, M$, MECT assigns request $i$ to the machine that provides the earliest expected completion time.

### D. Minimum Expected Execution Time (MEET)

The Minimum Expected Execution time (MEET) heuristic (based on the Minimum Execution Time heuristic, e.g., presented in [16], [25]) also ignores the robustness of each allocation. Instead, MEET allocates each request to its minimum expected execution time machine. That is, for each request $i$ select the machine $j$ that satisfies

$$j = \text{argmin}_{1 \le k \le M} \left( \mathbb{E}[r_{ik}] \right). \quad (2)$$

### E. K-Percent Best (KPB)

The K-percent best (KPB) heuristic [16] limits the number of machines that are considered at each request assignment to the $k$-percent of the machines with the shortest expected execution times. Using the expected value of the execution time of request $i$ on each machine $j = 1, \ldots, M$, identify the $k$-percent of the machines that provide the shortest expected request execution times. Next, calculate the expected completion time for request $i$ on each machine in the set of $k$-percent machines found previously and assign $i$ to the machine that provides the earliest expected completion time. For our simulations with eight machines, a value of $k$ equal to 37.5% (i.e., three of the eight machines) provided the best result.

### F. Shortest Queue (SQ)

In SQ, each request $i$ is assigned to the machine $j$ that has the smallest number of pending requests in its input queue. Ties are broken arbitrarily.

## V. Simulation Setup

Our simulation environment consisted of eight machines that exhibited inconsistent heterogeneous performance [25]; e.g., machine $A$ may be better than machine $B$ for application 1 but not for application 2. To model the sample mean application execution times on each machine, we used the base execution results for the 12 SPECint 2006 benchmark applications [27].[1] The mean execution times of the selected benchmarks were used to define the mean of a gamma distribution. Using these distributions, we generated 500 random sample execution times for each application on each machine [28] 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 [6] to the result to produce a noisy approximation of the original probability distributions—one for each application on each machine. Each benchmark served as a model for an application to be executed by the system creating an eight machine by twelve application matrix of execution time pmfs.

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 all of the machines for the application associated with the request.

To evaluate the effectiveness of our heuristics we conducted 100 different simulation trials. The frequency of request arrivals during each simulation is such that none of the heuristics tried were able to complete all of the requests by their assigned deadline for each trial. Each simulation trial included 2000 requests that arrived over a period of 20,000 time-steps. Request arrivals were assumed to follow a Poisson process, where the application associated with the request was selected at random from the application types available with a uniform probability.

[1]The eight machines chosen to compose the HC suite in our simulation trials were: Dell Precision 380 3Ghz Pentium Extreme Edition, Apple iMac 2Ghz Intel Core Duo, Apple XServe 2Ghz Intel Core Duo, IBM System X 3455 AMD Opteron 2347, Shuttle SN25P AMD Athlon 64 FX-60, IBM System P 570 4.7Ghz, SunFire 3800, and IBM BladeCenter HS21XM.

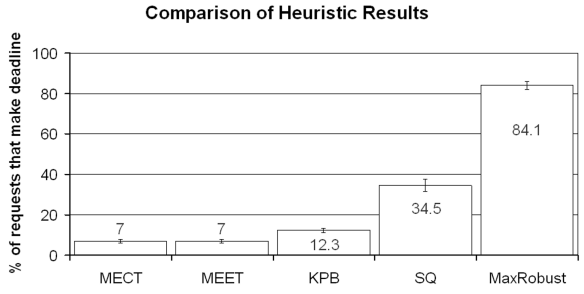


Fig. 1. A comparison of our heuristic results over 100 simulation trials. The results achieved for each heuristic are plotted along with their 95% confidence intervals.

## VI. Results

The heuristics were evaluated based on the percentage of applications that completed by their assigned deadlines. In evaluating our results, we observed that the number of on-time request completions that occur at the beginning (i.e., within the first 50 request arrivals) of each simulation trial tended to be high for all of the heuristics tried. At the beginning of each simulation trial, all of the machines in the HC suite are idle and more likely to complete requests on-time. That is, during this period the system is effectively under-subscribed. To limit the influence of simulation start up on our results, we chose to exclude the results for the first 50 requests in each simulation trial.

The results for all of the heuristics are plotted in Figure 1. In our simulations, the MaxRobust heuristic significantly outperformed all of the other heuristics tried, completing on average 84.1% of the applications by their deadline with a 95% confidence interval of $\pm$ 1.5%. The SQ heuristic completed 34.5% of the applications by their deadline with a 95% confidence interval of $\pm$ 2.8%. The KPB heuristic completed on average 12.3% of the applications on time with a 95% confidence interval of $\pm$ 0.6%. Both the MEET and MECT heuristics performed poorly, each completing on average 7% of the applications by their deadline both with a 95% confidence interval of $\pm$ 0.1%. The overall performance difference between the MaxRobust heuristic and the other approaches tried is significant and demonstrates the value of the stochastic robustness model for making allocation decisions.

The MECT and MEET results appear very similar with both only completing on average 7% of the requests on time. The poor performance of these heuristics is caused by poor request assignments early in the simulation trials that cause the on-time completion rate to drop significantly for the remainder of the simulation. Given the machines chosen for the simulations, one machine has a consistently higher expected execution time for all tasks in the simulation. Consequently, the MEET heuristic under-utilizes poor performing machines even if they could be used to complete specific tasks by their deadline.

In contrast, the MECT heuristic uses all of the machines in the HC suite. But it still ignores the possible variability in task execution times. Recall that the objective of each heuristic is to maximize the number of requests that complete by their deadline. The MECT heuristic allocates requests based on the expected application execution times. If there is a large variance between the actual execution time of an application and its expected execution time, then this may impact not only the results of the first request but also the results of the remaining requests in the input queue of the same machine. That is, because the completion time of each request is dependent on the completion times of requests previously assigned to the same machine, any unexpected increase in completion time can cause a cascade of failures. The problem is further compounded by the arrival rate of requests for processing, i.e., there is no large break in the arrival rate that would allow MECT to "catch up."

The KPB heuristic is a combination of MECT and MEET and significantly outperforms both. That is, KPB avoids some of the poor assignment problems of MECT by limiting the set of machines considered for each request assignment to only the $k$-percent of the machines (3 in these simulations) with the shortest expected execution times. Similarly, by accounting for the expected completion time of each request KPB is able to avoid the delays incurred by MEET.

Figure 2 plots the empirical cumulative distribution of the results for all of the heuristics. The x-axis of the figure defines the number of requests that miss their deadline. For each x, the y-axis plots the fraction of trials where x *or fewer* applications missed their deadline. The best result achievable in the figure corresponds to the line $y = 100\%$ (i.e., for all simulation trials, no requests miss their deadline). Comparing the results of the heuristics to this known limit on performance, we see that the MaxRobust heuristic consistently outperforms all of the other heuristics tried.

Surprisingly, the SQ heuristic performed significantly better than the KPB, MEET, and MECT heuristics. For example, the results for the SQ heuristic in Figure 2 are far better than the KPB, MECT, and MEET heuristics. That is, in 80% of the simulation trials SQ was able to complete 500 (out of 2000) requests by their deadline (i.e., no more than 1500 requests miss their deadline) whereas there were 0% of the trials where KPB, MECT, or MEET could complete that number of requests on-time. The KPB, MECT, and MEET heuristics suffer from the same effect, that is, if some request $A$ takes longer than expected, then a potentially large number of requests (waiting in that machine's input queue behind $A$) may miss their deadline. SQ addresses this effect by trying to minimize the maximum number of requests in the input queue of any machine. Selecting the machine with the shortest queue length lessens the impact of one or more tasks having a longer than expected completion time. That is, picking the shortest queue will in general lead to a balanced (equal) number of requests across all machines. Furthermore, by minimizing the max queue length, the accumulation of small increases in completion times for requests assigned to the same machine can be avoided.

To demonstrate the practicality of stochastic resource allocation in a dynamic environment, we timed several completion time calculations on an inexpensive Graphics Processing Unit (GPU). The most time-consuming calculation in computing a stochastic completion time is the convolution of the application execution time pmfs.

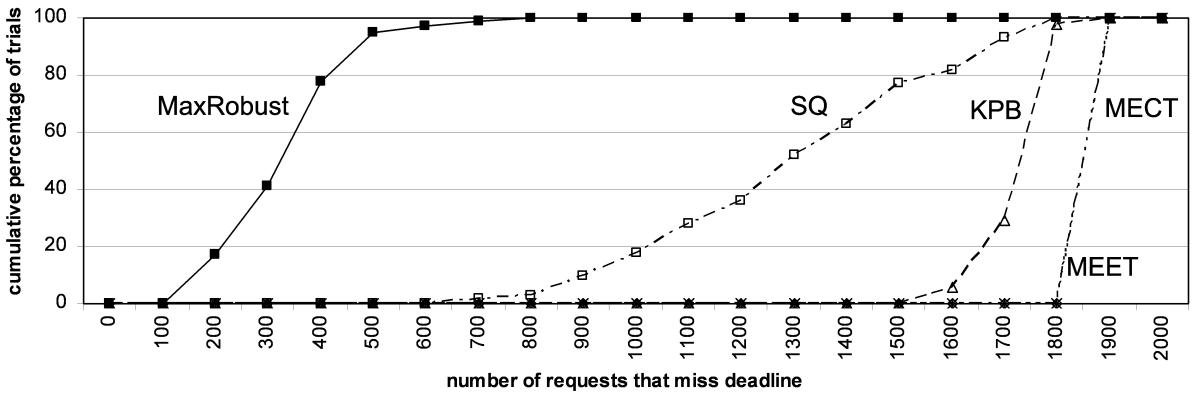


Fig. 2. A comparison of the results of the five heuristics where each curve represents the cumulative distribution for the 100 simulation trials corresponding to the number of requests that miss their deadline (plotted on the x-axis) relative to the percentage of trials where this number of requests or fewer miss their deadline (plotted on the y-axis).

The convolution calculation relies on the use of a discrete fast Fourier transform (DFFT). To accelerate this process we utilized the CUFFT package from NVIDIA [29] to compute the forward and inverse DFFTs on the GPU. To further minimize the amount of data transfer to and from the GPU we also computed the convolution on the GPU as well.

Using data from our simulations, the time to compute each completion time calculation on the GPU[2] was on average 0.0029 seconds. To determine this average we executed 10,000 convolutions where each distribution included 4096 entries. Thus, relative to the average execution time of each application in our simulated environment (which was on the order of tens of seconds) the time required to complete eight of these convolutions (one for each machine in the HC suite) to produce each assignment was trivial.

## VII. Related Work

According to the literature, the problem of workload distribution considered in our research falls into the category of dynamic resource allocation. The general problem of dynamically allocating a class of independent tasks to heterogeneous computing systems was studied in [16]. The primary objective in [16] 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 work: complete each request before its deadline. For comparison, we employed several heuristics from [16] in this environment and demonstrated the superior performance of the MaxRobust heuristic at achieving our resource allocation objective.

The robustness requirement in this work differs substantially from our earlier work on robustness in a dynamic environment in [30]. There, the robustness requirement was expressed in terms of the overall resource allocation, i.e., expressed in terms of the entire allocation. In this work, each application has an individual deadline, thus, the robustness metric must be expressed in terms of individual applications. Further, the research presented in [30] was not based on a stochastic model of application execution times.

In our previous work [3], similar to the environment considered in the current paper, each dynamically arriving task is assigned its own deadline relative to its arrival time and the execution time of each task is modeled as a random variable. However, our previous work focused on predicting the performance of heuristics in a stochastic dynamic environment.

In England et al. [31], the authors present a new robustness metric based on the Kolmogorov-Smirnov (K-S) statistic. Computing the K-S statistic for a resource allocation requires 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. In this case, however, the authors use the magnitude of the K-S statistic to measure the deviation of a system from its expected behavior. To characterize the overall robustness of a system, the authors measure the perturbed performance of the system assuming a number of different levels of perturbation. This technique for calculating robustness is well suited to the evaluation of a policy in advance of its deployment. However, the K-S statistic is not well suited to environments where robustness is to be calculated and used during resource allocation.

In Shi et al. [32], 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. In their system, task execution times are estimated and actual 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 the count of the number of schedule executions whose actual makespan is greater than the expected

[2]The GPU used was the Nvidia NVS 135M.

makespan. Neither approach to calculating robustness in [32] is based on stochastic information.

## VIII. Conclusions

In this work, we used a model of stochastic robustness that facilitates its calculation and use during resource allocation. 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 MaxRobust heuristic showed significant promise for achieving this desired result in a 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.

## 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] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *Journal of Parallel and Distributed Computing*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.

[3] 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 21$^{st}$ International Parallel and Distributed Processing Symposium (IPDPS 2007)*, Mar. 2007.

[4] S. Ali, A. A. Maciejewski, and H. J. Siegel, *Perspectives on Robust Resource Allocation for Heterogeneous Parallel Systems*. Boca Raton, FL: Chapman & Hall/CRC Press, 2008, pp. 41–1–41–30.

[5] 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.

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

[7] D. E. Irwin, L. E. Grit, and J. S. Chase, "Balancing risk and reward in a market-based task service," in *Proceedings of the 13$^{th}$ IEEE International Symposium on High Performance Distributed Computing (HPDC '04)*, June 2004, pp. 160–169.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.

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

[16] 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.

[17] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach," *Journal of Parallel and Distributed Systems*, vol. 47, no. 1, pp. 8–22, Nov. 1997.

[18] 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.

[19] 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.

[20] 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.

[21] B. A. Forouzan, *Data Communications and Networking*, 4th ed. New York, NY: McGraw-Hill Science, 2006.

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

[23] C. L. Phillips, J. M. Parr, and E. A. Riskin, *Signals, Systems, and Transforms*. Upper Saddle River, NJ: Pearson Education, 2003.

[24] 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.

[25] 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.

[26] V. Yarmolenko, J. Duato, D. K. Panda, and P. Sadayappan, "Characterization and enhancement of dynamic mapping heuristics for heterogeneous systems," in *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW '00)*, Aug. 2000, pp. 437–444.

[27] (2006) Standard performance evaluation corporation. Accessed Feb. 2006. [Online]. Available: http://www.spec.org/

[28] 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 50$^{th}$ Anniversary Issue*, vol. 3, no. 3, pp. 195–207, Nov. 2000.

[29] (2008) Cufft library. Accessed Jan. 2, 2009. [Online]. Available: http://developer.download.nvidia.com/compute/cuda/2_0/docs/CUFFT_Library_2.0.pdf

[30] 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*, vol. 42, no. 1, pp. 33–58, Oct. 2007.

[31] D. England, J. Weissman, and J. Sadagopan, "A new metric for robustness with application to job scheduling," in *Proceedings of*

*the 14$^{th}$ IEEE High Performance Distributed Computing (HPDC 05)*, Jul. 2005, pp. 135–143.

[32] 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.