ROBUST RESOURCE ALLOCATION IN HETEROGENEOUS PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS

INTRODUCTION

In parallel and distributed computing, multiple computers are collectively used to process a set of tasks simultaneously to improve performance over that of a single processor (1). Often, such computing systems are constructed from a heterogeneous mixture of machines that may differ in their capabilities (e.g., available memory, number of floating point units, clock speed, and operating system). In a heterogeneous computing system, the execution time of a task may differ depending on which computer executes the task. Often, task resource requirements lead to inconsistent

Wiley Encyclopedia of Computer Science and Engineering, edited by Benjamin Wah.

performance differences between heterogeneous machines. That is, machine 1 being faster than machine 2 on some task A does not imply that machine 1 is uniformly faster on all tasks.

Resource allocation in heterogeneous parallel and distributed computing is the process of assigning tasks to computers for execution such that some performance objective is optimized. For example, a common objective in resource allocation is to minimize the total time required to complete a set of tasks to be executed. It has been shown that resource allocation is an NP-hard problem (2) (i.e., an optimal solution cannot be found in reasonable time for problems of realistic size). Therefore, the task of resource allocation is often addressed heuristically. A resource allocation heuristic generates tasks to machine assignments that attempt to optimize the identified performance objective. The design of heuristics for resource allocation is an active area of research (3, 4). In this article, we will evaluate the robustness of resource allocations in both static and dynamic environments. In a static environment, the entire collection of tasks to be allocated is known in advance, prior to the start of allocation. In contrast, in a dynamic environment, the set of tasks to be executed is not known in advance and tasks are assigned as they arrive to the system.

Resource allocation decisions are often based on estimated values of task and system parameters, whose actual values are uncertain and may differ from available estimates. A resource allocation can be considered "robust" if it can mitigate the impact of uncertainties in system parameters on a given performance objective (5). That is, a robust resource allocation can guarantee a certain level of *performance* under a wide range of conditions. Any claim of robustness for a given system must answer these three questions (6): (*1*) What behavior makes the system robust? (*2*) What are the uncertainties that the system is robust against? (*3*) Quantitatively, exactly how robust is the system? These three questions help establish an intuitive meaning for the robustness of a system that goes beyond a simple nebulous adjective.

In the next section, we describe the formal FePIA procedure for deriving a robustness metric for any given system. Then we provide an example robustness metric derivation for environments where uncertainties are caused by system parameters whose values are only estimates. We use the steps of the FePIA procedure to define a model of robustness in a stochastic environment where uncertainties are modeled as random variables. We conclude with a brief discussion of the open problems in robust resource allocation.

DETERMINING A ROBUSTNESS METRIC

The three robustness questions provide the basis for the more formal FePIA procedure for deriving a quantitative measure of robustness (5). The procedure uses the following four steps for measuring the impact of uncertainty in estimated system parameters on a stated performance objective: (*1*) identify the performance features of interest within the system, (*2*) identify the source of uncertainty within the system (perturbation parameters), (*3*) clarify the impact of the system uncertainty on the performance features of interest, and (*4*) analyze the system to quantify robustness.

To illustrate the intuition behind the steps of the FePIA procedure, we will use the following simple resource allocation example throughout our discussion. In this example, a set of tasks is to be assigned to a heterogeneous collection of machines such that the finishing time of the last to finish machine (i.e., the total time required to complete all of the tasks, often referred to as makespan), is minimized. An estimate of the execution time of each task on each machine is available, and the resource allocation must take into account that there are unknown errors in these estimates (e.g., their actual values may be data dependent). The following summary outlines the four steps of the FePIA procedure, which was first presented in Ref. 5.

Step 1: Describe quantitatively the requirement that makes the system robust. This step provides a more precise formulation of the first of our intuitive robustness questions (question (*1*) in the previous section). Based on this robustness requirement, we identify the performance features of the system that determine whether the robustness requirement is met. Establishing the acceptable variation in performance features requires that we define the limits on these features that allow us to maintain an acceptable level of performance. For example, the acceptable variation in makespan for our sample system may be to limit the actual makespan to some constant value τ. That is, the actual finish time of the last-to-finish machine should be less than or equal to τ.

Step 2: Identify the uncertainties in system parameters whose values may impact the performance features that are to be limited in variation (question (*2*) in the previous section). The uncertainties in estimated system parameter values are referred to as the perturbation parameters of the system. We are interested in the perturbation parameters that may cause a variation in the performance features of interest (i.e., those identified in Step 1 as part of the robustness requirement). For our makespan example, the performance metric is based on estimates of task execution times, and these estimates may contain unknown errors that could impact system performance. That is, the uncertainty in task execution times are relevant because changes in these values may directly impact the makespan of the system. Thus, we are interested in the robustness of the estimated makespan relative to unknown errors in task execution time estimates.

Step 3: Identify the impact of perturbation parameters (identified in Step 2) on the performance features of the system (identified in Step 1). With respect to our robustness requirement from Step 1, the actual value of our performance feature must be within the identified acceptable level of variation. For the makespan example, the sum of the *actual* execution times for all tasks assigned to any given machine determines the *actual* finishing time of that machine. Thus, the actual finishing time of the

last-to-finish machine (i.e., the actual makespan) must be less than or equal to τ. Differences between the *estimated* task execution times and their actual values will directly impact the ability of the system to meet the robustness requirement established in Step 1.

Step 4: The last step is to conduct an analysis to determine the smallest collective change in the assumed values of the perturbation parameters of Step 2 that would cause any of the performance features of Step 1 to violate their robustness requirements. The value produced by this analysis will provide the degree of robustness for the system (addressing question (*3*) of the previous section). For the makespan example, this is a quantification of the smallest collective increase in task execution times that would lead to the actual makespan being greater than τ.

DETERMINISTIC MODELS OF ROBUSTNESS

Introduction

In this section, we provide two example derivations of a robustness metric, one in a static environment and one in a dynamic environment. For both environments, we present an example heuristic that uses the derived robustness metric for that environment during resource allocation to facilitate the creation of robust resource allocations.

Example Static Environment

Deriving a Robustness Metric. In this environment, similar to the makespan example of the previous section, the goal of the resource allocation is to assign T tasks to M machines such that the robustness of the system is maximized (7). Because this is a static environment, the T tasks to be assigned are all known in advance. Furthermore, we assume that the estimated time to compute (ETC) each task on each machine has been provided [determined by experimental or analytical techniques (8)]. We also assume that the M machines are heterogeneous (i.e., each ETC value for a given task on the M machines may be different). Using the FePIA procedure, we can quantitatively define a measure of robustness for this example system as follows (Table 1 provides a reference for the terms defined in this section).

Step 1: We first describe the robustness requirement. For this system, we require that the actual finishing time of each machine be less than or equal to a fixed constant τ.

Step 2: Uncertainty in this system originates because of unknown inaccuracies in the estimates of task execution times, which can lead directly to increases in machine finishing times that may violate our robustness requirement. For our example system, unknown inaccuracies in the ETC values are expected (e.g., the actual execution time of a task may be data dependent). Thus, the perturbation parameters for our system are these inaccuracies, and we require that resource allocations in this environment be robust to these inaccuracies.

Table 1. Table of Term Definitions

T	tasks to be assigned
M	total number of heterogeneous machines
ETC	matrix of estimated task execution times
τ	resource allocation finishing time constraint
μ	resource allocation
C^{est}	vector of estimated task execution times
C	vector of actual task execution times
$F_j(C^{est}, \mu)$	estimated finishing time of machine j
$r_\mu(F_j(C^{est}, \mu), C)$	robustness radius of machine j
ϕ	performance features
$\rho_\mu(\phi, C)$	robustness metric
$\mathcal{T}_j$	subset of tasks assigned to machine j

Step 3: To understand the impact that uncertainties in ETC times can have on machine finishing times, we need to define a model for calculating machine finishing times in this system. For a given resource allocation μ, let C^{est} be the vector of estimated execution times for the T tasks to be executed, and let C be the corresponding vector of actual execution times for the tasks (i.e., C^{est} plus the estimation error for each execution time). The finishing time for each machine j ($1 \leq j \leq M$) is determined based on the execution times for tasks assigned to that machine under a specified resource allocation μ. Given C^{est}, we can denote the estimated finishing time of machine j under resource allocation μ, as $F_j(C^{\text{est}}, \mu)$. Let $\mathcal{T}_j$ be the subset of tasks in T assigned to machine j under resource allocation μ. We can calculate the estimated finishing time of each machine j as the sum of the task execution times for all tasks in $\mathcal{T}_j$. The set of performance features of interest for the system, denoted ϕ, are the set of actual finishing times for the machines given our resource allocation μ (i.e., $\phi = \{F_j(C, \mu) 1 \leq j \leq M\}$). Therefore, the unknown errors in our ETC estimates will directly impact the performance features of interest within this system.

Step 4: Finally, we conduct an analysis to determine exactly how robust the system is under a specific resource allocation. The robustness radius of a performance feature, denoted $r_{\mu(F_j(C^{\text{est}}, \mu), C)}$, is defined as the smallest collective increase in system parameters that would lead to a violation of the robustness requirement for that performance feature. That is, for a given machine j, we would like to know the smallest collective increase in the execution times of tasks assigned to that machine that would result in $F_j(C, \mu) > \tau$. Quantitatively, if the Euclidean distance between the vector of actual computation times and the vector of estimated computation times for tasks assigned to machine j is no larger than $r_\mu(F_j(C^{\text{est}}, \mu), C)$, then the finishing time of machine j will be less than the makespan constraint τ. Because the finishing time of a machine is the sum of the execution times of all tasks assigned to that machine, the makespan constraint can be represented as a hyperplane in a multidimensional space whose axes are defined for each machine in terms of the tasks assigned it. That is, each perturbation parameter provides a single dimension along which the robustness radius

can vary. For example, in Fig. 1, the two tasks c_1 and c_2 have been assigned to machine j and provide the axes for this geometric analysis. The radius $r_\mu(F_j(C^{\text{est}}, \mu), C)$, interpreted geometrically, is the shortest distance from C^{est} to the hyperplane given by $F_j(C^{\text{est}}, \mu) = \tau$. Thus, as long as the estimation error contained in C^{est} does not exceed $r_\mu(F_j(C^{\text{est}}, \mu))$, then the finishing time of machine j will not exceed τ. In Fig. 1, the robustness requirement is plotted as a solid gray line—highlighting the boundary between robust performance and nonrobust performance. The estimated execution times of c_1 and c_2 are plotted together in the figure as the estimated value of the perturbation parameters. Figure 1 demonstrates the robustness radius for this example as the shortest distance from the point estimate of the perturbation parameters to the hyperplane defined by the robustness requirement (i.e., a line in two dimensions). The general calculation of $r_\mu(F_j(C^{\text{est}}, \mu), C)$ can be expressed using the point-to-plane distance formula as follows:

$$r_\mu(F_j(C^{\text{est}}, \mu), C) = \frac{\tau - F_j(C^{\text{est}}, \mu)}{\sqrt{\text{number of tasks assigned to machine } j}} \quad (1)$$

Finally, the robustness metric can be expressed in terms of the set of robustness radii, where the metric is equal to the smallest of the robustness radii for the set of performance features ϕ, denoted ρ_μ. Mathematically, this is expressed as follows,

$$\rho_\mu(\phi, C) = \min_{F_j(C^{\text{est}}, \mu)\ \phi} r_\mu(F_j(C^{\text{est}}, \mu), C) \quad (2)$$

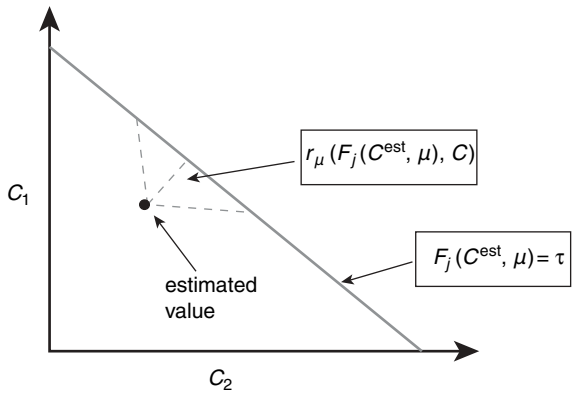

Figure 1. An example geometric analysis of the robustness radius for a given machine. In the example, resource allocation μ includes the assignment of tasks c_1 and c_2 to a single machine j. The robustness requirement for the finishing time of this machine [i.e., $F_j(C^{\text{est}}, \mu) = \tau$] can be interpreted as a hyperplane (a line in two dimensions). The estimated execution times for tasks c_1 and c_2 define a point within the space of all possible values for the perturbation parameters of this system. The robustness radius for this machine under allocation μ is the shortest distance from the point estimate of the perturbation parameters to the hyperplane defining the robustness requirement for this machine.

As long as the collective increase in execution times for the set of tasks assigned to a given machine does not exceed the robustness metric value, then the system will continue to meet the robustness requirement.

Using Static Robustness. The robustness metric can be used directly to compare resource allocations for their ability to deliver on promised performance, as in Ref. 5. Alternatively, the robustness metric can be used during resource allocation to improve the overall robustness of the resulting allocation. That is, using the robustness metric from the previous subsection, we can define a resource allocation heuristic that attempts to maximize the robustness of the resource allocations that it produces. In this subsection, we give an example resource allocation heuristic that attempts to maximize robustness.

The max–max heuristic, presented in Ref. 7, uses a measure of robustness during allocation to maximize robustness. Max–max starts with an initial set of tasks to be assigned to machines within the system. For each task, we consider what the robustness radius for each machine would be if that task were assigned to it and select the machine that provides the largest overall robustness radius. From this set of task-machine pairs, max–max selects the task-machine pair that provides the overall largest robustness radius. The selected task is assigned to its selected machine and removed from the set of tasks to be assigned; then the machine's completion time is updated accordingly. Max–max continues in this way until all tasks have been assigned to machines in the heterogeneous system. Note that when evaluating $r_\mu(F_j(C^{\text{est}}, \mu), C)$, as in Eq. (1), during resource allocation we replace the complete machine finishing time $F_j(C^{\text{est}})$ with the finishing time of the machine given the set of tasks that have previously been assigned to this machine plus the task under consideration. This value can be thought of as an intermediate robustness radius determined by the partial allocation when it is calculated.

This heuristic was evaluated in Ref. 7 alongside several other resource allocation techniques. By performing resource allocation in this way, the resulting allocation can tolerate a larger variation in task execution times than many commonly used techniques, as demonstrated in Ref. 7. The paper also demonstrates the utility of this approach through comparison with some other techniques for resource allocation.

Example Dynamic Environment

Deriving a Robustness Metric. This subsection focuses on deriving a robustness metric for a dynamic resource allocation environment, where the set of tasks to execute and their arrival times are not known in advance. The set of tasks to be executed in this environment is assumed to be taken from a frequently executed collection of tasks, as is common in many environments. Consequently, the ETC values for these tasks on each of the machines in the computing environment are assumed to be known. Although the ETC times are known for the tasks to be executed, the actual execution times may vary because of a

dependence on the characteristics of the input data that are not known until execution time.

Tasks in a dynamic environment can be assigned in either immediate mode or batch mode. In immediate mode, tasks are assigned immediately as they arrive. In batch mode, tasks are instead collected as they arrive and assigned as a batch (9, 10). The robustness metric developed in this subsection is appropriate for use in heuristics that operate in either immediate mode or batch mode. In the next subsection, we will give an example of an immediate mode heuristic developed for this environment.

In this dynamic environment, T independent tasks arrive dynamically, where the set of tasks to execute and their arrival times are not known in advance. Each arriving task is assigned to one machine within the set of M heterogeneous machines. The robustness of a resource allocation must be determined at each mapping event, where mapping events occur whenever a task arrives or completes. Note that because this is a time-varying problem, the relevant set of tasks to consider is time dependent. Let $T(t)$ be the set of tasks at time t whose arrival time is less than or equal to t and have not completed execution by time t. Let $F_j(t)$ be the predicted finishing time of machine j, at time t, for a given resource allocation μ, based on the provided ETC values. Let $MQ_j(t)$ denote the subset of $T(t)$ previously mapped to machine j, and let $scet_j(t)$ denote the start time of the currently executing task on machine j. Using these parameters, we can mathematically express $F_j(t)$ as follows:

$$F_j(t) = scet_j(t) + \sum_{\forall i\, MQ_j(t)} \mathrm{ETC}(i,j) \tag{3}$$

According to the first section, we can intuitively define robustness by considering answers to the three questions of that section. What behavior makes the system robust? To be robust, the finishing time of each machine at each mapping event should be limited in variation. What uncertainties is the system robust against? In this environment, unknown estimation errors in the ETC values may cause machine finishing times to increase unpredictably. Quantitatively, exactly how robust is the system? To answer this question, we have to determine exactly how much variation in task execution times can be tolerated while ensuring that the finishing time of each machine at each mapping event remains within a allowed range. We address these points more precisely using the FePIA procedure to derive a robustness metric for this environment. Table 2 provides a comprehensive list of term definitions for this section.

Step 1: We first define the robustness requirement for this system in terms of $F_j(t)$ (i.e., the performance feature of interest is the maximum of the machine finishing times at each mapping event). Let $\beta(t)$ denote the maximum of the *predicted* machine finishing times at time t. That is,

$$\beta(t) = \max_{\forall j\, M} (F_j(t)) \tag{4}$$

A resource allocation is considered robust if at each mapping event, the *actual* finishing time for each machine is no more than τ seconds greater than $\beta(t)$ [i.e., $\forall j, F_j(t) \leq \tau + \beta(t)$].

Step 2: In this environment, task execution time estimates are subject to unknown estimation errors that may cause actual task execution times to deviate from their predicted values. Thus, the perturbation parameter of this system is the uncertainty in task execution time estimates.

Step 3: The identified perturbation parameter will directly impact the $F_j(t)$ values. That is, because $F_j(t)$ is found using the sum of the ETC values for all tasks in $MQ_j(t)$, any increase in the execution time of a task in $MQ_j(t)$ over its estimate can cause $F_j(t)$ to increase.

Step 4: Given a resource allocation μ at time t, the robustness radius $r_{\mu(F_j(t))}$ of machine j can be defined as the largest collective increase in the estimated task execution times that can occur without violating the robustness requirement. Given the count of the number of tasks assigned to machine j at time t, expressed as $MQ_j(t)$, and using the point-to-plane formula of the previous subsection, we can express $r_\mu(F_j(t))$ as follows:

$$r_\mu(F_j(t)) = \frac{\tau + \beta(t) - F_j(t)}{\sqrt{MQ_j(t)}} \tag{5}$$

In the static environment of the previous subsection, the number of robustness radii was equivalent to the number of machines in the computing system. However, in this environment, we need to measure the robustness radius of each machine at each point in time where our information about the robustness radius may change. Let $\rho_{\mu(t)}$ be the robustness of the resource allocation as measured at time t and found as follows:

$$\rho_\mu(t) = \min_{\forall j\, M} r_\mu(F_j(t)) \tag{6}$$

Thus, because the mixture of tasks pending execution changes whenever a task arrives and all tasks must

Table 2. Table of Term Definitions

$T(t)$	set of tasks to be assigned at time t
$F_j(t)$	predicted finishing time of machine j at time t
$MQ_j(t)$	subset of tasks previously mapped to machine j at time t
$scet(j)$	start time of the currently executing task on machine j
$\beta(t)$	maximum of the predicted machine finishing times at time t
$r_\mu(F_j(t))$	robustness radius of the completion time of machine j at time t
$\rho_\mu(t)$	robustness metric value at time t

complete, there will be $2 \times T \times M$ robustness radii to be considered. The minimum over all of these robustness radii will provide the robustness metric, denoted ρ_μ, for the resource allocation. Let E be the set of all times when mapping events occur in the system during a resource allocation. Mathematically, ρ_μ can be expressed as follows:

$$\rho_\mu = \min_{\forall e \, E} \rho_\mu(e) \tag{7}$$

Defining the robustness metric in this way, ρ_μ corresponds to the largest collective deviation from assumed circumstances that the resource allocation can tolerate while ensuring that system performance will remain acceptable. In particular, in this example system, ρ_μ corresponds to the largest collective increase in task execution times that the system can tolerate and still guarantee that at all mapping events $F_j(t) \leq \beta(t) + \tau$.

Using Dynamic Robustness. This subsection uses the example problem formulation of the previous subsection (presented in Ref. 10 and focuses on generating a dynamic resource allocation for a set of dynamically arriving, independent tasks. The resource allocation is expected to minimize $\beta(t)$, while being able to tolerate a quantifiable amount of variation in the ETC values for the assigned tasks. Therefore, the goal of heuristics in this environment is to assign tasks to machines such that βt is minimized at each mapping event while maintaining a specified level of robustness (e.g., $\rho_\mu(t) \geq \alpha$ at each mapping event). The following example of an immediate mode heuristic, known as feasible k-percent best, successfully addresses these competing concerns by iteratively reducing the set of machines under consideration until a "best" machine has been selected.

Because this resource allocation heuristic operates in immediate mode, each task is assigned as it arrives. For each newly arrived task, feasible k-percent best identifies the set of all "feasible" machines for the task. The set of feasible machines is defined for each task when it arrives and includes only those machines that will satisfy the robustness requirement for the system even if the task under consideration were assigned to it [i.e., $r_\mu(F_j(t)) \geq \alpha$]. Thus, it reduces the set of machines under consideration to only those that would satisfy the robustness constraint. If no machines are feasible, then the heuristic must exit with an error condition indicating that no allocation exists given the current circumstances that can satisfy the robustness requirement.

Recall that this resource allocation environment employs a heterogeneous collection of machines; thus, the possible execution times for a given task may vary from one machine to the next. To reduce the set of machines under consideration further, we select from the set of feasible machines the k machines that would provide the smallest execution times for the task under consideration. Intuitively, we want to minimize the impact that this task execution has on future task completion times by ensuring that the execution time of this task is relatively small. Thus, we select a subset of the feasible machines that consists of only the k machines that provide the smallest execution times for the task. For these machines, we compute the completion time for each of the machines and assign the task to the machine that provides the smallest completion time for the task. The heuristic continues in this way until either an error occurs or the resource allocation is terminated.

STOCHASTIC MODELS OF ROBUSTNESS

Introduction

In some environments, more information may be available regarding the probability of variations in system parameters. For example, we may have historical information regarding past execution times for a given task that can be used to approximate the probabilities of all possible execution times. This stochastic information can be used to derive a robustness metric. In a stochastic environment, we model the system parameters that contain uncertainty as random variables and we assume that stochastic information is available that characterizes this uncertainty beyond a simple point estimate, as used in the previous section.

To illustrate the usefulness of stochastic data in resource allocation, consider an allocation environment with two machines (A and B). In the example situation of Fig. 2, some tasks have previously been assigned to machines A and B. We would like to select a machine to execute task 3, and we would like task 3 to complete by the plotted completion time constraint. If we use a point estimate for the completion time of task 3 (e.g., the mean of the completion time distributions), then it would seem that the best decision would be to assign task 3 to machine A whose point estimate of the completion time is smaller than on machine B. However, by using the full stochastic information, we can see that although the point estimate of the completion time

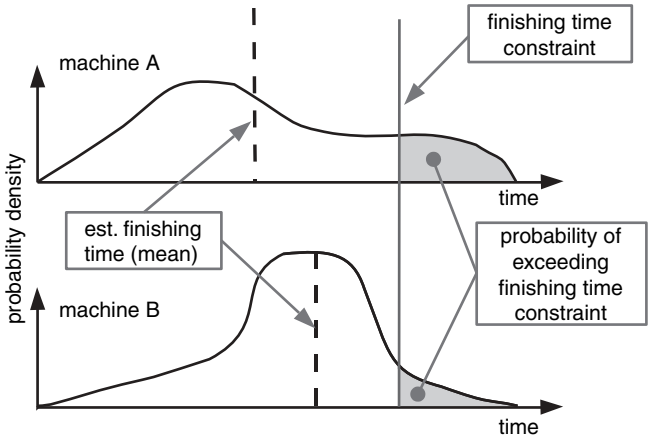

Figure 2. An example system where we are to assign task 3 to either machine A or machine B, and we would like task 3 to complete prior to the plotted completion time constraint. Presented are the task completion time probability density functions for task 3 on both machine A and machine B. Using the point estimate, plotted as a dashed line in the figure, machine A seems to be the better choice. However, accounting for the complete stochastic information describing all possible completion times, machine B has a lower probability to violate the makespan constraint and is clearly the better choice.

distribution for task 3 on machine A is smaller than on machine B, the tail of the distribution is much smaller on B than on A. That is, there is a much higher probability that task 3 will violate its completion time constraint on machine A, making machine B the statistically better choice.

Stochastic Robustness in a Static Environment

Deriving a Robustness Metric. In this environment, we are concerned with allocating a set of T tasks to M heterogeneous machines where we are concerned with system makespan as the performance metric. Below, we use the FePIA procedure to derive a robustness metric for this environment. Table 3 provides a comprehensive list of term definitions for this section.

Step 1: For this system, the performance feature of interest is system makespan, denoted ψ. A resource allocation can be considered robust if the actual finishing time of each machine is less than or equal to a fixed constant $\beta_{\max}$ (i.e., $\psi \leq \beta_{\max}$).

Step 2: Uncertainty in this system originates because the exact execution time for each task is unknown. We can model the execution time of each task i on each machine j as a random variable (11), denoted η_{ij}.

Step 3: The finishing time of each machine in a stochastic environment is calculated as the sum of the execution time random variables for each task assigned to that machine (12). Let n_j be the count of the number of tasks assigned to machine j. The finishing time of machine j, referred to as a local performance characteristic ψ_j, can be expressed as follows:

$$\psi_j = \sum_{i=1}^{n_j} \eta_{ij} \tag{8}$$

Thus, the system makespan can be expressed in terms of the local performance characteristics as follows:

$$\psi = \max\{\psi_1, \ldots, \psi_M\} \tag{9}$$

Because of its functional dependence on the execution time random variables, the system makespan is a random variable. That is, the uncertainty in task execution times can have a direct impact on the performance metric of this system.

Step 4: Finally, we conduct an analysis to determine exactly how robust the system is under a specific resource allocation. The stochastic robustness metric, denoted θ, is defined as the probability that the performance characteristic of the system is less than or equal to $\beta_{\max}$ (i.e., $\theta = \mathbb{P}[\psi \leq \beta_{\max}]$). For a given resource allocation, the stochastic robustness metric measures the probability that the generated system performance will satisfy our robustness requirement. Clearly, unity is the most desirable stochastic robustness metric value (i.e., there is a zero probability that the system will violate the established robustness requirement.)

Assuming no intertask data transfers exist among the tasks to be assigned, the random variables for the local performance characteristics $(\psi_1, \psi_2, \ldots, \psi_M)$ are mutually independent. As such, the stochastic robustness metric for a resource allocation can be found as the product of the probability that each local performance feature is less than or equal to $\beta_{\max}$. Mathematically, this is given as follows:

$$\theta = \prod_{\forall j} (\mathbb{P}[\psi_j \leq \beta_{\max}]) \tag{10}$$

If the execution times η_{ij} for tasks assigned to machine j are mutually independent, then the summation of Eq. (8) can be computed using an $(n_j\text{-}1)$-fold convolution of the corresponding pmfs (12,13).

Table 3. Table of Term Definitions

ψ	performance feature
$\beta_{\max}$	finishing time constraint
η_{ij}	execution time random variable of task i on machine j
ψ_j	finishing time of machine j
θ	stochastic robustness metric

Using Static Robustness. Two ways of using the static stochastic robustness metric are apparent by inspecting the parameters of Eq. (10). The two parameters $\beta_{\max}$ and θ can alternately be either optimized or specified by the user. For example, the user could specify a $\beta_{\max}$ value as a constraint and employ a heuristic to attempt to maximize the robustness (θ) of the resulting resource allocation. That is, in this case, the user is interested in a resource allocation that has the highest probability to complete all of the tasks by $\beta_{\max}$.

Maximizing the robustness of a resource allocation given a fixed $\beta_{\max}$ requires minimizing the ψ_j values, thus, maximizing the probability that each machine will finish before $\beta_{\max}$.

For some systems, it may be unclear how to select an appropriate $\beta_{\max}$ value. Thus, we can instead define a minimum acceptable θ value, denoted ω, and attempt to minimize $\beta_{\max}$ such that the probability that all tasks completed by $\beta_{\max}$ is at least ω. In this subsection, we consider the case where the minimum acceptable robustness value ω is specified by the user and we want to minimize $\beta_{\max}$ such that $\theta \geq \omega$. In Ref. 14, the period minimization routine (*PMR*) was introduced to iterate through the possible $\beta_{\max}$ values for a *given resource allocation* to find the smallest $\beta_{\max}$ value, provided by that allocation, such that $\theta \geq \omega$.

To demonstrate the use of the stochastic robustness metric in a resource allocation, we introduce a static stochastic environment where a set of machines periodically receive data sets to be processed by a collection of n tasks (14). Each dataset must be processed by all of the tasks before the next dataset arrives. The execution times for each of the n tasks is assumed to be inherently data dependent; thus, the exact execution time for each task is unknown prior to its execution. However, we are provided a pmf describing the probabilities of task execution times for each machine.

It is important to note that because this is a static environment, we are determining the allocation of tasks to machines *in advance* of deploying the system, (i.e., before it will be required to process real data). Thus, by optimizing the allocation of machines to tasks in advance of processing real data, we can improve the frequency with which the system can process datasets.

The stochastic robustness metric for this system is the probability that the actual makespan of the system does not exceed $\beta_{\max}$ (i.e., $\theta = \mathbb{P}[\psi \leq \beta_{\max}]$). The goal of resource allocation heuristics in this environment is to minimize $\beta_{\max}$ such that θ is always greater than or equal to a given fixed probability ω (i.e., $\theta \geq \omega$). Intuitively, by specifying a minimum robustness value (θ), the user is specifying an acceptable probability for the makespan to be greater than $\beta_{\max}$ (i.e., 1-θ).

We can use this formulation of robustness along with the PMR procedure to design a two-phase greedy heuristic for resource allocation in this system (14). The two-phase greedy heuristic first initializes the set of tasks to be executed to the entire set of tasks that are available. Although there are still tasks to execute, the heuristic uses two phases to find the next task assignment. In the first phase, the heuristic determines a machine assignment for each unassigned task that minimizes $\beta_{\max}$ (ignoring all other unassigned tasks), where we use the PMR procedure to determine the smallest $\beta_{\max}$ for each possible allocation such that the robustness constraint (ω) is still satisfied. In the second phase, the heuristic selects the task machine pair (found in the first phase) that provides the smallest overall $\beta_{\max}$. The selected task is then allocated to its chosen machine and removed from the set of tasks to be assigned. The heuristic continues in this way until all tasks have been allocated.

This subsection presented just two example uses of the static stochastic robustness metric; many more uses may be possible. The next section considers open problems in robust resource allocation and heterogeneous computing as a whole.

OPEN PROBLEMS

The open problems in robust resource allocation are directly related to the long-term goals of heterogeneous computing research (15). The principal goal in heterogeneous computing (HC) research is to develop software environments that automatically assign and execute applications, where applications are expressed in a machine-independent, high-level language. Developing such environments will facilitate the use of heterogeneous computing by (*1*) increasing software portability (i.e., programmers need not be concerned with the machine details of the heterogeneous environment) and (*2*) increasing the possibility of deriving better task machine assignments than users themselves derive using ad hoc methods.

A four-stage conceptual model for using a heterogeneous computing environment composed of dedicated machines is shown in Fig. 3, based on Ref. 16. The rectangles in the figure indicate actions to be taken, and the ovals indicate the information produced by those actions. The model is "conceptual" because as yet no complete automatic implementation exists. The goal of the model is to describe the steps required for the automatic execution of applications in a heterogeneous computing environment. Each of the rectangles in Fig. 3 represents an open research problem, where additional new techniques need to be developed.

In stage 1, the system will determine parameters relevant to both the applications that are to be executed and the machines that are to execute them. The information generated by this stage includes a scheme for categorization of application needs and a similar scheme for machine capabilities. An application is assumed to be composed of one or more independent tasks. Some tasks can again be further decomposed into a collection of two or more communicating subtasks, where subtasks are assumed to have data dependencies (e.g., execution results may need to be communicated between subtasks). It is assumed that individual subtasks may be assigned to different machines for execution.

The information generated in stage 1 is passed to stage 2, where task profiling and analytical benchmarking are performed. In task profiling, applications are partitioned into tasks and subtasks that have different computational needs, where the computational needs within a given task are consistent. Each of the tasks and subtasks is then profiled to determine quantitatively their computational requirements. Analytical benchmarking quantifies the performance of each machine in the suite with respect to executing each type of operation being considered. Techniques for performing task profiling and analytical benchmarking are needed.

In stage 3, task profiling data and analytical benchmarking results are combined to create execution time estimates for each task and subtask on each machine in the suite. These results, along with initial loading and "status" of each machine in the suite, are used to generate machine assignments, based on a chosen optimization criteria, for each task and each subtask to be executed. Hierarchical scheduling techniques are of interest to allow the development of very large HC environments (e.g., grids) (17). In some environments, the scale of the distributed system requires that task assignment be done in a distributed fashion, as opposed to a centralized resource allocation.

Stage 4 corresponds to the execution of the applications in the heterogeneous computing environment. In general, information regarding task execution times may be estimated in advance (e.g., taken from the task profiling of stage 2), but exact information regarding actual task execution times may not be known in advance (e.g., task execution times may be data dependent). In a dynamic environment, task completion times and machine loading/status information are monitored during execution and may be used to influence resource allocation decisions. For example, the information may be used to alter task machine assignments to reflect current user needs. Improved methods for determining and disseminating the current loading and status of machines in the HC suite must also be determined. Similarly, methods are required for monitoring current network load and status.

To realize this automatic heterogeneous computing environment requires further research in many areas. For example, in stage 2, machine-independent languages

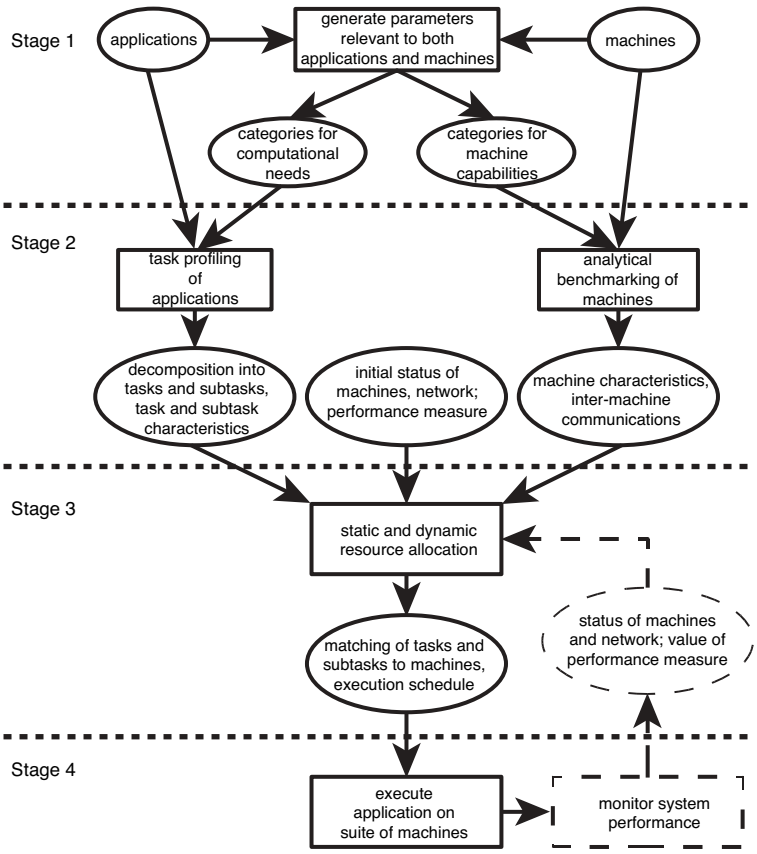

Figure 3. A model of required support for using heterogeneous computing environments. Ovals indicate information and rectangles actions. The dashed lines represent the components needed to perform a dynamic resource allocation.

with user-specified directives are needed to (*1*) allow compilation of applications into efficient code suitable for any machine in an HC system, (*2*) aid in decomposing applications into tasks and subtasks, and (*3*) facilitate the determination of task and subtask computational requirements.

Incorporating a model for multitasking within a machine is another area of ongoing HC research related to stage 2. Most modern operating systems support some level of multitasking for an individual processor; however, it is unclear how to incorporate this information into the resource allocation process.

Another area of research related to stage 2 involves modeling uncertainty in perturbation parameters for robust resource allocation. For example, in a stochastic environment, methods are needed for leveraging experiential data to model uncertainty in perturbation parameters (18). This is important because prior work in stochastic resource allocation environments has assumed that these models are available (i.e., in the form of probability mass functions). Furthermore, once a pmf has been established for a perturbation parameter, methods are needed for updating the existing pmf with new experiential data. Updating pmfs with the most current information is important in a dynamic environment because it enables the model to track changes in perturbation parameter distributions over time. In addition, pmfs based on experiential data may provide only an approximation of the true distribution of perturbation parameter values, and methods are needed for determining the impact of estimation error in pmfs on robustness calculations. That is, how to make resource allocation decisions robust with respect to estimation errors in perturbation parameter pmfs is an open problem.

Many HC environments have inherent quality-of-service constraints that must be met during resource allocation. Determining such resource allocations is an active area of research related to stage 3. For example, this article has presented several examples of HC environments with quality-of-service constraints and has shown how to apply the robustness methodology to determine resource allocations that meet those constraints. In addition to these examples, other quality-of-service requirements might involve multiple robustness requirements (e.g., minimum bandwidth requirements), guaranteed processor time for certain users, or real-time response capabilities.

Current research in robust resource allocation, related to stage 3, is investigating combining multiple types of perturbation parameters into a single robustness metric (e.g., combining machine failure probabilities and task execution time uncertainty into a single metric). Combining multiple

perturbation parameters into a single measure of robustness will extend the applicability of robust resource allocation into problem domains where these uncertainties occur simultaneously.

In a stochastic environment, resource allocation decisions made in stage 3 of our model may depend on combining perturbation parameter pmfs. For example, pmfs for perturbation parameters can be used to produce an overall metric for the robustness of a resource allocation. In a dynamic environment, where resource allocation decisions must be made quickly, it is important to identify new fast methods for combining perturbation parameter distributions to produce a robustness value during resource allocation. If heuristics can quickly combine perturbation parameter distributions, then we can determine methods for using the stochastic robustness metric to guide resource allocation decisions during execution.

SUMMARY

Robust resource allocation is an important research area within heterogeneous parallel and distributed computing systems. The three robustness questions are fundamental to the understanding of robustness in any system: (*1*) What behavior makes the system robust? (*2*) What uncertainties must the system be robust against? (*3*) Quantitatively, exactly how robust is the system? These three core questions led to the development of the FePIA procedure for deriving a robustness metric. We have applied the FePIA procedure to derive robustness metrics in a variety of contexts, including several different perturbation parameters. In addition to demonstrating the derivation of a robustness metric, we have also demonstrated many ways to incorporate a robustness metric into resource allocation heuristics.

ACKNOWLEDGMENTS

The authors would like to thank Chris Klumph, Kody Willman, and Luis Briceño for their valuable comments. This research was supported by the NSF under Grant CNS-0615170 and by the Colorado State University George T. Abell Endowment.

BIBLIOGRAPHY

1. 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, *J. Parallel Distr. Comput.*, **61** (6): 810–837, 2001.
2. O. H. Ibarra and C. E. Kim, Heuristic algorithms for scheduling independent tasks on non-identical processors, *J. ACM*, **24** (2): 280–289, 1977.
3. L. Bölöni and D. Marinescu, Robust scheduling of metaprograms, *J. Scheduling*, **5** (5): 395–412, 2002.
4. A. Dogan and F. Ozguner, Genetic Algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems, *Cluster Comput.*, **7** (2): 177–190, 2004.
5. S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, Measuring the robustness of a resource allocation, *IEEE Trans. Parallel Dist. Syst.*, **15** (7): 630–641, 2004.
6. S. Ali, A. A. Maciejewski, and H. J. Siegel, *Perspectives on Robust Resource Allocation for Heterogeneous Parallel Systems*, Boca Raton, FL: Chapman & Hall, 2008, pp. 411–4130.
7. P. Sugavanam, H. J. Siegel, A. A. Maciejewski, M. Oltikar, A. Mehta, R. Pichel, A. Horiuchi, V. Shestak, M. Al-Otaibi, Y. Krishnamurthy, S. Ali, J. Zhang, M. Aydin, P. Lee, K. Guru, M. Raskey, and A. Pippin, Robust static allocation of resources for independent tasks under makespan and dollar cost constraints, *J. Parallel Dist. Comput.*, **67** (4): 400–416, 2007.
8. 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, *J. Parallel Dist. Comput.*, **44** (1): 35–52, 1997.
9. 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, *J. Parallel Dist. Comput.*, **59** (2): 107–121, 1999.
10. 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, *J. Supercomput.*, **42** (1): 33–58, 2007.
11. L. Wasserman, *All of Statistics: A Concise Course in Statistical Inference*, New York: Springer, 2005.
12. V. Shestak, J. Smith, H. J. Siegel, and A. A. Maciejewski, A stochastic approach to measuring the robustness of resource allocations in distributed systems, *Proc. of the 2006 International Conference on Parallel Processing (ICPP 2006)*, 2006 pp. 6–12.
13. A. Leon-Garcia, *Probability & Random Processes for Electrical Engineering*, Reading, MA: Addison Wesley, 1989.
14. 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, *Proc. of the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications*, 2006, pp. 3–9.
15. A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. Wang, Heterogeneous computing: Challenges and opportunities, *IEEE Comput.*, **26** (6): 18–27, 1993.
16. M. Maheswaran, T. D. Braun, and H. J. Siegel, Heterogeneous distributed computing, in *Encyclopedia of Electrical and Electronics Engineering*, New York: Wiley, 1999.
17. I. Foster, and C. Kesselman, (eds.), *The Grid 2: Blueprint for a New Computing Infrastructure*, San Francisco, CA: Morgan Kaufmann, 1999.
18. M. A. Iverson, F. Ozguner, and L. Potter, Statistical prediction of task execution times through analytical benchmarking for scheduling in a heterogeneous environment, *IEEE Trans. Comput.*, **48** (12): 1374–1379, 1999.

JAY SMITH
HOWARD JAY SIEGEL
ANTHONY A. MACIEJEWSKI
Colorado State University
Fort Collins, Colorado