# Measuring the Robustness of a Resource Allocation

Shoukat Ali, *Member*, *IEEE*, Anthony A. Maciejewski, *Senior Member*, *IEEE*, Howard Jay Siegel, *Fellow*, *IEEE*, and Jong-Kook Kim, *Student Member*, *IEEE*

**Abstract**—Parallel and distributed systems may operate in an environment that undergoes unpredictable changes causing certain system performance features to degrade. Such systems need robustness to guarantee limited degradation despite fluctuations in the behavior of its component parts or environment. This research investigates the robustness of an allocation of resources to tasks in parallel and distributed systems. The main contributions of this paper are 1) a mathematical description of a metric for the robustness of a resource allocation with respect to desired system performance features against multiple perturbations in multiple system and environmental conditions, and 2) a procedure for deriving a robustness metric for an arbitrary system. For illustration, this procedure is employed to derive robustness metrics for three example distributed systems. Such a metric can help researchers evaluate a given resource allocation for robustness against uncertainties in specified perturbation parameters.

**Index Terms**—Robustness, robustness metric, resource allocation, resource management systems, parallel and distributed systems.

## 1 Introduction

Parallel and distributed systems may operate in an environment where certain system performance features degrade due to unpredictable circumstances, such as sudden machine failures, higher than expected system load, or inaccuracies in the estimation of system parameters (e.g., [3], [4], [13], [16], [17], [18], [20]). An important question then arises: Given a system design, what extent of departure from the assumed circumstances will cause a performance feature to be unacceptably degraded? That is, how robust is the system? Before answering this question, one needs to clearly define robustness. Robustness has been defined in different ways by different researchers. According to [17], robustness is the degree to which a system can function correctly in the presence of inputs different from those assumed. In a more general sense, [13] states that a robust system continues to operate correctly across a wide range of operational conditions. Robustness, according to [16], guarantees the maintenance of certain desired system characteristics despite fluctuations in the behavior of its component parts or its environment. The concept of robustness, as used in this research, is similar to that in [16]. Like [16], this work emphasizes that robustness should be defined for a given set of system features, with a given set of perturbations applied to the system. This research investigates the robustness of resource allocation in parallel and distributed systems and accordingly customizes the definition of robustness.

Parallel and distributed computing is the coordinated use of different types of machines, networks, and interfaces to meet the requirements of widely varying application mixtures and to maximize the system performance or cost-effectiveness. An important research problem is how to determine a resource allocation (which may involve the matching of applications to resources and ordering their execution) so as to maximize robustness of desired system features against perturbations. This research addresses the design of a robustness metric for resource allocations.

A resource allocation is defined to be *robust with respect to specified system performance features against perturbations in specified system parameters* if degradation in these features is limited when the perturbations occur. For example, if a resource allocation has been declared to be robust with respect to satisfying a throughput requirement against perturbations in the system load, then the system configured under that allocation should continue to operate without a throughput violation when the system load increases. The immediate question is: What is the *degree* of robustness? That is, for the example given above, how much can the system load increase before a throughput violation occurs? This research addresses this question and others related to it by formulating the mathematical description of a metric that evaluates the robustness of a resource allocation with respect to certain system performance features against multiple perturbations in multiple system components and environmental conditions. In addition, this work outlines a procedure called FePIA (named after the four steps that constitute the procedure) for deriving a robustness metric for an arbitrary system. For illustration, the procedure is employed to derive robustness

- *S. Ali is with the Department of Electrical and Computer Engineering, University of Missouri-Rolla, Rolla, MO 65409-0040. E-mail: shoukat@umr.edu.*
- *A.A. Maciejewski is with the Department of Electrical and Computer Engineering, Colorado State University, Fort Collins, CO 80523-1373. E-mail: aam@colostate.edu.*
- *H.J. Siegel is with the Department of Electrical and Computer Engineering and the Department of Computer Science, Colorado State University, Fort Collins, CO 80523-1373. E-mail: hj@colostate.edu.*
- *J.-K. Kim is with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907-1285. E-mail: jongkook@purdue.edu.*

*Manuscript received 23 Jan. 2003; revised 10 Sept. 2003; accepted 20 Oct. 2003.*
*For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 118184.*

TABLE 1
Glossary of Notation

| | |
|---|---|
| $\Phi$ | the set of all performance features |
| $\phi_i$ | the $i$-th element in $\Phi$ |
| $\langle \beta_i^{\min}, \beta_i^{\max} \rangle$ | a tuple that gives the bounds of the tolerable variation in $\phi_i$ |
| $\Pi$ | the set of all perturbation parameters |
| $\boldsymbol{\pi}_j$ | the $j$-th element in $\Pi$ |
| $n_{\boldsymbol{\pi}_j}$ | the dimension of vector $\boldsymbol{\pi}_j$ |
| $\mu$ | a resource allocation |
| $r_\mu(\phi_i,\ \boldsymbol{\pi}_j)$ | the robustness radius of resource allocation $\mu$ with respect to $\phi_i$ against $\boldsymbol{\pi}_j$ |
| $\rho_\mu(\Phi,\ \boldsymbol{\pi}_j)$ | the robustness of resource allocation $\mu$ with respect to set $\Phi$ against $\boldsymbol{\pi}_j$ |
| $\mathcal{A}$ | the set of applications |
| $\mathcal{M}$ | the set of machines |
| $\mathbf{P}$ | a weighted concatenation of the vectors $\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \cdots, \boldsymbol{\pi}_{\lvert\Pi\rvert}$ |

metrics for three example distributed systems. The robustness metric and the FePIA procedure for its derivation are the main contributions of this paper.

The rest of the paper is organized as follows: Section 2 describes the FePIA procedure mentioned above. It also defines a generalized robustness metric. Derivations of this metric for three example parallel and distributed systems are given in Section 3. Section 4 extends the definition of the robustness metric given in Section 2 to multiple specified perturbation parameters. The computational complexity of the robustness metric calculation is addressed in Section 5. Section 6 presents some experiments that highlight the usefulness of the robustness metric. A sampling of the related work is given in Section 7. Section 8 concludes the paper. A glossary of the notation used in this paper is given in Table 1. Note that, throughout this paper, new symbols are underlined when they are introduced. Such underlining is not a part of the symbology.

## 2 Generalized Robustness Metric

This section proposes a general procedure, called *FePIA*, for deriving a general robustness metric for any desired computing environment. The name for the above procedure stands for identifying the performance *fe*atures, the *p*erturbation parameters, the *i*mpact of perturbation parameters on performance features, and the *a*nalysis to determine the robustness. Specific examples illustrating the application of the FePIA procedure to sample systems are given in the next section. Each step of the FePIA procedure is now described.

1. Describe quantitatively the requirement that makes the system robust. Based on this *robustness requirement*, determine the QoS performance features that should be limited in variation to ensure that the robustness requirement is met. Identify the acceptable variation for these feature values as a result of uncertainties in system parameters. Consider an example where

    a. the QoS performance feature is *makespan* (the total time it takes to complete the execution of a set of applications) for a given resource allocation,
    b. the acceptable variation is up to 30 percent of the makespan that was calculated for the given resource allocation using estimated execution times of applications on the machines they are assigned, and
    c. the uncertainties in system parameters are inaccuracies in the estimates of these execution times.

    Mathematically, let $\underline{\Phi}$ be the set of system performance features that should be limited in variation. For each element $\phi_i \in \Phi$, quantitatively describe the tolerable variation in $\phi_i$. Let $\left\langle \underline{\beta_i^{\min}}, \underline{\beta_i^{\max}} \right\rangle$ be a tuple that gives the bounds of the tolerable variation in the system feature $\phi_i$. For the makespan example, $\phi_i$ is the time the $i$th machine finishes its assigned applications, and its corresponding $\langle \beta_i^{\min}, \beta_i^{\max} \rangle$ could be $\langle 0,\ 1.3 \times\ (\text{estimated makespan value}) \rangle$.
2. Identify all of the system and environment parameters whose values may impact the QoS performance features selected in Step 1. These are called the *perturbation parameters* (these are similar to hazards in [4]), and the performance features are required to be robust with respect to these perturbation parameters. For the makespan example above, the resource allocation (and its associated predicted makespan) was based on the estimated application execution times. It is desired that the makespan be robust (stay within 130 percent of its estimated value) with respect to uncertainties in these estimated execution times.

    Mathematically, let $\underline{\Pi}$ be the set of perturbation parameters. It is assumed that the elements of $\Pi$ are vectors. Let $\pi_j$ be the $j$th element of $\Pi$. For the makespan example, $\pi_j$ could be the vector composed of the actual application execution times, i.e., the $i$th element of $\pi_j$ is the actual execution time of the $i$th application on the machine it was assigned. In general, representation of the perturbation parameters as separate elements of $\Pi$ would be based on their nature or kind (e.g., message length variables in $\pi_1$ and computation time variables in $\pi_2$).

3. Identify the impact of the perturbation parameters in Step 2 on the system performance features in Step 1. For the makespan example, the sum of the actual execution times for all of the applications assigned a given machine is the time when that machine completes its applications. Note that Step 1b implies that the actual time each machine finishes its applications must be within the acceptable variation.

   Mathematically, for every $\phi_i \in \Phi$, determine the relationship $\phi_i = f_{ij}(\boldsymbol{\pi}_j)$, if any, that relates $\phi_i$ to $\boldsymbol{\pi}_j$. In this expression, $f_{ij}$ is a function that maps $\boldsymbol{\pi}_j$ to $\phi_i$. For the makespan example, $\phi_i$ is the finishing time for machine $m_i$, and $f_{ij}$ would be the sum of execution times for applications assigned to machine $m_i$. The rest of this discussion will be developed assuming only one element in $\Pi$. The case where multiple perturbation parameters can affect a given $\phi_i$ simultaneously will be examined in Section 4.

4. The last step is to determine the smallest collective variation in the values of perturbation parameters identified in Step 2 that will cause any of the performance features identified in Step 1 to violate its acceptable variation. This will be the degree of robustness of the given resource allocation. For the makespan example, this will be some quantification of the total amount of inaccuracy in the execution times estimates allowable before the actual makespan exceeds 130 percent of its estimated value.

   Mathematically, for every $\phi_i \in \Phi$, determine the *boundary values of* $\boldsymbol{\pi}_j$, i.e., the values satisfying the *boundary relationships* $f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min}$ and $f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}$. (If $\boldsymbol{\pi}_j$ is a discrete variable, then the boundary values correspond to the closest values that bracket each boundary relationship. See Section 3.3 for an example.) These relationships separate the region of robust operation from that of nonrobust operation. Find the smallest perturbation in $\boldsymbol{\pi}_j$ that causes any $\phi_i \in \Phi$ to exceed the bounds $\langle \beta_i^{\min}, \beta_i^{\max} \rangle$ imposed on it by the robustness requirement.

   Specifically, let $\underline{\boldsymbol{\pi}_j^{\text{orig}}}$ be the value of $\boldsymbol{\pi}_j$ at which the system is originally assumed to operate. However, due to inaccuracies in the estimated parameters or changes in the environment, the value of the variable $\boldsymbol{\pi}_j$ might differ from its assumed value. This change in $\boldsymbol{\pi}_j$ can occur in different "directions" depending on the relative differences in its individual components. Assuming that no information is available about the relative differences, all values of $\boldsymbol{\pi}_j$ are possible. Fig. 1 illustrates this concept for a single feature, $\phi_i$, and a two-element perturbation vector $\boldsymbol{\pi}_j \in \mathbf{R}^2$. The curve shown in Fig. 1 plots the set of boundary points $\{\boldsymbol{\pi}_j | f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}\}$ for a resource allocation $\underline{\mu}$. For this figure, the set of boundary points $\{\boldsymbol{\pi}_j | f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min}\}$ is given by the points on the $\pi_{j1}$-axis and $\pi_{j2}$-axis.

   The region enclosed by the axes and the curve gives the values of $\boldsymbol{\pi}_j$ for which the system is robust with respect to $\phi_i$. For a vector $\mathbf{x} = [x_1\ x_2\ \cdots\ x_n]^{\mathrm{T}}$, let $\underline{\|\mathbf{x}\|_2}$ be the $\ell_2$-norm (Euclidean norm) of the vector, defined by $\sqrt{\sum_{r=1}^{n} x_r^2}$. The point on the curve marked as $\underline{\boldsymbol{\pi}_j^*(\phi_i)}$ has the property that the Euclidean distance from $\boldsymbol{\pi}_j^{\text{orig}}$ to $\boldsymbol{\pi}_j^*(\phi_i)$, $\|\boldsymbol{\pi}_j^*(\phi_i) - \boldsymbol{\pi}_j^{\text{orig}}\|_2$, is the smallest over all such distances from $\boldsymbol{\pi}_j^{\text{orig}}$ to a point on the curve. An important interpretation of $\boldsymbol{\pi}_j^*(\phi_i)$ is that the value $\|\boldsymbol{\pi}_j^*(\phi_i) - \boldsymbol{\pi}_j^{\text{orig}}\|_2$ gives the largest Euclidean distance that the variable $\boldsymbol{\pi}_j$ can change in *any* direction from the assumed value of $\boldsymbol{\pi}_j^{\text{orig}}$ without the performance feature $\phi_i$ exceeding the tolerable variation. Let the distance $\|\boldsymbol{\pi}_j^*(\phi_i) - \boldsymbol{\pi}_j^{\text{orig}}\|_2$ be called the *robustness radius*, $\underline{r_\mu(\phi_i, \boldsymbol{\pi}_j)}$, of $\phi_i$ against $\boldsymbol{\pi}_j$. Mathematically,

$$r_\mu(\phi_i, \boldsymbol{\pi}_j) = \min_{\boldsymbol{\pi}_j:\ (f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}) \vee (f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min})} \|\boldsymbol{\pi}_j - \boldsymbol{\pi}_j^{\text{orig}}\|_2. \tag{1}$$

   *This work defines $r_\mu(\phi_i, \boldsymbol{\pi}_j)$ to be the robustness of resource allocation $\mu$ with respect to performance feature $\phi_i$ against the perturbation parameter $\boldsymbol{\pi}_j$.*

   The robustness definition can be extended easily for all $\phi_i \in \Phi$. It is simply the minimum of all robustness radii. Mathematically, let

$$\rho_\mu(\phi, \boldsymbol{\pi}_j) = \min_{\phi_i \in \Phi} \big( r_\mu(\phi_i, \boldsymbol{\pi}_j) \big). \tag{2}$$

   Then, $\rho_\mu(\phi, \boldsymbol{\pi}_j)$ is the *robustness metric of resource allocation $\mu$ with respect to the performance feature set $\Phi$ against the perturbation parameter $\boldsymbol{\pi}_j$.*

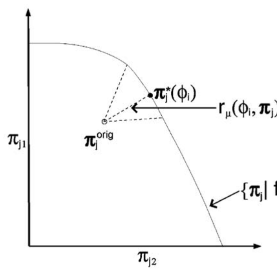

$\pi_{j1}$
$\pi_{j2}$
$\boldsymbol{\pi}_j^{\text{orig}}$
$\boldsymbol{\pi}_j^*(\phi_i)$
$r_\mu(\phi_i, \boldsymbol{\pi}_j)$
$\{\boldsymbol{\pi}_j | f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}\}$

Fig. 1. Some possible directions of increase of the perturbation parameter $\boldsymbol{\pi}_j$ and the direction of the smallest increase. The curve plots the set of points, $\{\boldsymbol{\pi}_j | f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}\}$. The set of boundary points, $\{\boldsymbol{\pi}_j | f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min}\}$ is given by the points on the $\pi_{j1}$-axis and $\pi_{j2}$-axis.

Even though the $\ell_2$-norm has been used for the robustness radius in this general formulation, in practice, the choice of a norm should depend on the particular environment for which a robustness measure is being sought. Section 3.3 gives an example situation where the $\ell_1$-norm is preferred over the $\ell_2$-norm.

In addition, in some situations, changes in some elements of $\boldsymbol{\pi}_j$ may be more probable than changes in other elements. In such cases, one may be able to modify the distance calculation so that the contribution from an element with a larger probability to change has a proportionally larger weight. This is a subject for future study.

## 3 Derivations of Robustness Metric for Example Systems

### 3.1 Independent Application Allocation System

The first example derivation of the robustness metric is for a system that allocates a set of independent applications to a set of machines [6]. In this system, it is required that the makespan (defined as the completion time for the entire set of applications) be robust against errors in application execution time estimates. Specifically, the actual makespan under the perturbed execution times must be no more than a certain factor times the predicted makespan calculated using the assumed execution times. It is obvious that the larger the "factor," the larger the robustness. Assuming that $\ell_2$-norm is used, one might also reason that as the number of applications assigned to a given machine increases, the change in the finishing time for that machine will increase due to errors in the application computation times. As will be seen shortly, the instantiation of the general framework for this system does reflect this intuition.

A brief description of the system model is now given. The applications are assumed to be independent, i.e., no communications between the applications are needed. The set $\underline{\mathcal{A}}$ of applications is to be allocated to a set $\underline{\mathcal{M}}$ of machines so as to minimize the makespan (defined as the finishing time of the machine that finishes last). Each machine executes a single application at a time (i.e., no multitasking), in the order in which the applications are assigned. Let $\underline{C_{ij}}$ be the *estimated time to compute* (*ETC*) for application $\underline{a_i}$ on machine $\underline{m_j}$. It is assumed that $C_{ij}$ values are known for all $i$, $j$, and a resource allocation $\mu$ is determined using the ETC values. In addition, let $\underline{F_j}$ be the time at which $m_j$ finishes executing all of the applications allocated to it.

Assume that unknown inaccuracies in the ETC values are expected, requiring that the resource allocation $\mu$ be robust against them. More specifically, it is required that, for a given resource allocation, its actual makespan value $\underline{M}$ (calculated considering the effects of ETC errors) may be no more than $\tau$ times its *nominal value*, $\underline{M^{\text{orig}}}$. The nominal value of the makespan is the value calculated assuming the ETC values are accurate. Following Step 1 of the FePIA procedure in Section 2, the system performance features that should be limited in variation to ensure the makespan robustness are the finishing times of the machines. That is, $\Phi = \{F_j | 1 \leq j \leq |\mathcal{M}|\}$.

According to Step 2 of the FePIA procedure, the perturbation parameter needs to be defined. Let $\underline{C_i^{\text{orig}}}$ be the ETC value for application $a_i$ on the machine where it is allocated. Let $\underline{C_i}$ be equal to the actual computation time value ($\underline{C_i^{\text{orig}}}$ plus the estimation error). In addition, let $\boldsymbol{C}$ be the vector of the $C_i$ values such that $\boldsymbol{C} = [C_1 \ C_2 \ \cdots \ C_{|\mathcal{A}|}]$. Similarly, $\underline{\boldsymbol{C}^{\text{orig}}} = [\underline{C_1^{\text{orig}}} \ \underline{C_2^{\text{orig}}} \ \cdots \ \underline{C_{|\mathcal{A}|}^{\text{orig}}}]$. The vector $\boldsymbol{C}$ is the perturbation parameter for this analysis.

In accordance with Step 3 of the FePIA procedure, $F_j$ has to be expressed as a function of $\boldsymbol{C}$. To that end,

$$F_j(\boldsymbol{C}) = \sum_{i:\, a_i \text{ is allocated to } m_j} C_i. \tag{3}$$

Note that the finishing time of a given machine depends only on the actual execution times of the applications allocated to that machine and is independent of the finishing times of the other machines. Following Step 4 of the FePIA procedure, the set of boundary relationships corresponding to the set of performance features is given by $\{F_j(\boldsymbol{C}) = \tau M^{\text{orig}} | 1 \leq j \leq |\mathcal{M}|\}$.

For a two-application system, $\boldsymbol{C}$ corresponds to $\boldsymbol{\pi}_j$ in Fig. 1. Similarly, $C_1$ and $C_2$ correspond to $\pi_{j1}$ and $\pi_{j2}$, respectively. The terms $\boldsymbol{C}^{\text{orig}}$, $F_j(\boldsymbol{C})$, and $\tau M^{\text{orig}}$ correspond to $\boldsymbol{\pi}_j^{\text{orig}}$, $f_{ij}(\boldsymbol{\pi}_j)$, and $\beta_i^{\max}$, respectively. The boundary relationship "$F_j(\boldsymbol{C}) = \tau M^{\text{orig}}$" corresponds to the boundary relationship "$f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}$."

From (1), the robustness radius of $F_j$ against $\boldsymbol{C}$ is given by

$$r_\mu(F_j, \boldsymbol{C}) = \min_{\boldsymbol{C}:\, F_j(\boldsymbol{C}) = \tau M^{\text{orig}}} \|\boldsymbol{C} - \boldsymbol{C}^{\text{orig}}\|_2. \tag{4}$$

That is, if the Euclidean distance between any vector of the actual execution times and the vector of the estimated execution times is no larger than $r_\mu(F_j, \boldsymbol{C})$, then the finishing time of machine $m_j$ will be at most $\tau$ times the predicted makespan value.

Note that the right-hand side in (4) can be interpreted as the perpendicular distance from the point $\boldsymbol{C}^{\text{orig}}$ to the hyperplane described by the equation $\tau M^{\text{orig}} - F_j(\boldsymbol{C}) = 0$. Using the point-to-plane distance formula [21], (4) reduces to

$$r_\mu(F_j, \boldsymbol{C}) = \frac{\tau M^{\text{orig}} - F_j(\boldsymbol{C}^{\text{orig}})}{\sqrt{\text{number of applications allocated to } m_j}}. \tag{5}$$

The robustness metric, from (2), is $\rho_\mu(\Phi, \boldsymbol{C}) = \min_{F_j \in \Phi} r_\mu(F_j, \boldsymbol{C})$. That is, if the Euclidean distance between any vector of the actual execution times and the vector of the estimated execution times is no larger than $\rho_\mu(\phi, \boldsymbol{C})$, then the actual makespan will be at most $\tau$ times the predicted makespan value. The value of $\rho_\mu(\phi, \boldsymbol{C})$ has the units of $\boldsymbol{C}$, namely, time.

### 3.2 The HiPer-D System

The second example derivation of the robustness metric is for a HiPer-D [15] like system that allocates a set of continuously executing, communicating applications to a set of machines. It is required that the system be robust with respect to certain QoS attributes against unforeseen increases in the "system load."

The HiPer-D system model used here was developed in [1] and is summarized here for reference. The system consists of heterogeneous sets of sensors, applications, machines, and actuators. Each machine is capable of multitasking, executing the applications allocated to it in a round-robin fashion. Similarly, a given network link is multitasked among all data transfers using that link. Each sensor produces data periodically at a certain rate and the resulting data streams are input into applications. The applications process the data and send the output to other applications or to actuators. The applications and the data transfers between them are modeled with a directed acyclic graph, shown in Fig. 2. The figure also shows a number of *paths* (enclosed by dashed lines) formed by the applications. A *path* is a chain of producer-consumer pairs that starts at a sensor (the *driving sensor*) and ends at an actuator (if it is a

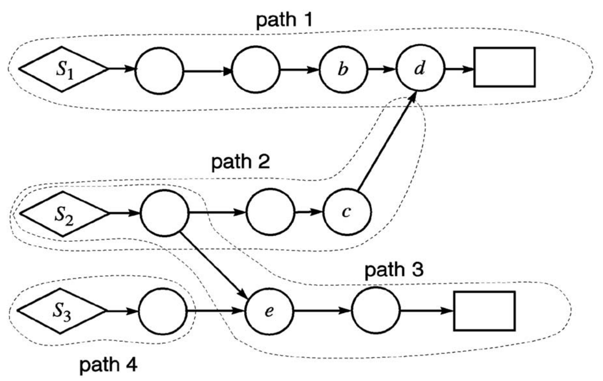


Fig. 2. The DAG model for the applications (circles) and data transfers (arrows). The diamonds and rectangles denote sensors and actuators, respectively. The dashed lines enclose each path formed by the applications.

"trigger path") or at a multiple-input application (if it is an "update path"). In the context of Fig. 2, path 1 is a trigger path and path 2 is an update path. In a real system, application $d$ could be a missile firing program that produces an order to fire. It needs target coordinates from application $b$ in path 1 and an updated map of the terrain from application $c$ in path 2. Naturally, application $d$ must respond to any output from $b$, but must not issue fire orders if it receives an output from $c$ alone; such an output is used only to update an internal database. So, while $d$ is a multiple input application, the rate at which it produces data is equal to the rate at which the "trigger" application $b$ produces data (in the HiPer-D model). That rate, in turn, equals the rate at which the driving sensor, $S_1$, produces data. The problem specification indicates the path to which each application belongs, and the corresponding driving sensor.

Let $\underline{\mathcal{P}}$ be the set of all paths, and $\underline{\mathcal{P}_k}$ be the list of applications that belong to the $k$th path. Note that an application may be present in multiple paths. As in Section 3.1, $\mathcal{A}$ is the set of applications.

The sensors constitute the interface of the system to the external world. Let the maximum periodic data output rate from a given sensor be called its *output data rate*. The *minimum throughput constraint* states that the computation or communication time of any application in $\mathcal{P}_k$ is required to be no larger than the reciprocal of the output data rate of the driving sensor for $\mathcal{P}_k$. For application $\underline{a_i} \in \mathcal{P}_k$, let $\underline{R(a_i)}$ be set to the output data rate of the driving sensor for $\mathcal{P}_k$. In addition, let $\underline{T_{ij}^c}$ be the computation time for application $\underline{a_i}$ allocated to machine $\underline{m_j}$. Also, let $\underline{T_{ip}^n}$ be the time to send data from application $a_i$ to application $a_p$. Because this analysis is being carried out for a specific resource allocation, the machine where a given application is allocated is known. It is assumed that $a_i$ is allocated to $m_j$ and the machine subscript for $T_{ij}^c$ is omitted in the ensuing analysis for clarity unless the intent is to show the relationship between execution times of $a_i$ at various possible machines.

The *maximum end-to-end latency* constraint states that, for a given path $\mathcal{P}_k$, the time taken between the instant the driving sensor outputs a data set until the instant the actuator or the multiple-input application fed by the path receives the result of the computation on that data set must be no greater than a given value, $\underline{L_k^{\max}}$. Let $\underline{L_k}$ be the actual (as opposed to the maximum allowed) value of the end-to-end latency for $\mathcal{P}_k$. The quantity $L_k$ can be found by adding the computation and communication times for all applications in $\mathcal{P}_k$ (including any sensor or actuator communications). Let $\underline{\mathcal{D}(a_i)}$ be the set of successor applications of $a_i$. Then,

$$L_k = \sum_{\substack{i:\, a_i \in \mathcal{P}_k \\ p:\, (a_p \in \mathcal{P}_k) \wedge (a_p \in \mathcal{D}(a_i))}} \left[ T_i^c + T_{ip}^n \right]. \tag{6}$$

It is desired that a given resource allocation $\mu$ of the system be robust with respect to the satisfaction of two QoS attributes: the latency and throughput constraints. Following Step 1 of the FePIA procedure in Section 2, the system performance features that should be limited in variation are the latency values for the paths and the computation and communication time values for the applications. The set $\Phi$ is given by

$$\begin{aligned}\Phi = &\{T_i^c \mid 1 \le i \le |\mathcal{A}|\} \\ &\bigcup \{T_{ip}^n \mid (1 \le i \le |\mathcal{A}|) \wedge (\text{for } p \text{ where } a_p \in \mathcal{D}(a_i))\} \\ &\bigcup \{L_k \mid 1 \le k \le |\mathcal{P}|\}.\end{aligned} \tag{7}$$

This system is expected to operate under uncertain outputs from the sensors requiring that the resource allocation $\mu$ be robust against unpredictable increases in the sensor outputs. Let $\underline{\boldsymbol{\lambda}_z}$ be the output from the $z$th sensor in the set of sensors, and be defined as the number of objects present in the most recent data set from that sensor. The *system workload*, $\underline{\boldsymbol{\lambda}}$, is the vector composed of the load values from all sensors. Let $\underline{\boldsymbol{\lambda}^{\text{orig}}}$ be the initial value of $\boldsymbol{\lambda}$, and $\lambda_i^{\text{orig}}$ be the initial value of the $i$th member of $\boldsymbol{\lambda}^{\text{orig}}$. Following Step 2, the perturbation parameter $\boldsymbol{\pi}_j$ is identified to be $\boldsymbol{\lambda}$.

Step 3 of the FePIA procedure requires that the impact of $\boldsymbol{\lambda}$ on each of the system performance features be identified. The computation times of different applications (and the communication times of different data transfers) are likely to be of different complexities with respect to $\boldsymbol{\lambda}$. Assume that the dependence of $T_i^c$ and $T_{ip}^n$ on $\boldsymbol{\lambda}$ is known (or can be estimated) for all $i, p$. Given that, $T_i^c$ and $T_{ip}^n$ can be reexpressed as functions of $\boldsymbol{\lambda}$ as $T_i^c(\boldsymbol{\lambda})$ and $T_{ip}^n(\boldsymbol{\lambda})$, respectively. Even though $d$ is triggered only by $b$, its computation time depends on the outputs from both $b$ and $c$. In general, $T_i^c(\boldsymbol{\lambda})$ and $T_{ip}^n(\boldsymbol{\lambda})$ will be functions of the loads from all those sensors that can be traced back from $a_i$. For example, the computation time for application $d$ in Fig. 2 is a function of the loads from sensors $S_1$ and $S_2$, but that for application $e$ is a function of $S_2$ and $S_3$ loads (but each application has just one driving sensor: $S_1$ for $d$ and $S_2$ for $e$). Then, (6) can be used to express $L_k$ as a function of $\boldsymbol{\lambda}$.

Following Step 4 of the FePIA procedure, the set of boundary relationships corresponding to (7) is given by

$$\begin{aligned}&\{T_i^c(\boldsymbol{\lambda}) = 1/R(a_i) \mid 1 \le i \le |\mathcal{A}|\} \bigcup \\ &\{T_{ip}^n(\boldsymbol{\lambda}) = 1/R(a_i) \mid (1 \le i \le |\mathcal{A}|\}) \wedge (\text{for } p \text{ where } a_p \in \mathcal{D}(a_i))\} \\ &\bigcup \{L_k(\boldsymbol{\lambda}) = L_k^{\max} \mid 1 \le k \le |\mathcal{P}|\}.\end{aligned}$$

Then, using (1), one can find, for each $\phi_i \in \Phi$, the robustness radius, $r_\mu(\phi_i, \boldsymbol{\lambda})$. Specifically,

$$r_\mu(\phi_i, \boldsymbol{\lambda}) = \begin{cases} \min\limits_{\boldsymbol{\lambda}:\ T_x^c(\boldsymbol{\lambda})=1/R(a_x)} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{orig}}\|_2 & \text{if } \phi_i = T_x^c \quad (8) \\ \min\limits_{\boldsymbol{\lambda}:\ T_{xy}^n(\boldsymbol{\lambda})=1/R(a_x)} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{orig}}\|_2 & \text{if } \phi_i = T_{xy}^n \quad (9) \\ \min\limits_{\boldsymbol{\lambda}:\ L_k=L_k^{\max}} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{orig}}\|_2 & \text{if } \phi_i = L_k. \quad (10) \end{cases}$$

The robustness radius in (8) is the largest increase (Euclidean distance) in load in any direction (i.e., for any combination of sensor load values) from the assumed value that does not cause a throughput violation for the computation of application $a_x$. This is because it corresponds to the value of $\boldsymbol{\lambda}$ for which the computation time of $a_x$ will be at the allowed limit of $1/R(a_x)$. The robustness radii in (9) and (10) are similar values for the communications of application $a_x$ and the latency of path $\mathcal{P}_k$, respectively. The robustness metric, from (2), is given by $\rho_\mu(\Phi, \boldsymbol{\lambda}) = \min_{\phi_i \in \Phi}\big(r_\mu(\phi_i, \boldsymbol{\lambda})\big)$. For this system, $\rho_\mu(\Phi, \boldsymbol{\lambda})$ is the largest increase in load in any direction from the assumed value that does not cause a latency or throughput violation for any application or path. Note that $\rho_\mu(\Phi, \boldsymbol{\lambda})$ has the units of $\boldsymbol{\lambda}$, namely, objects per data set. In addition, note that, although $\boldsymbol{\lambda}$ is a discrete variable, it has been treated as a continuous variable in (8) for the purpose of simplifying the illustration. A method for handling a discrete perturbation parameter is discussed in Section 3.3.

## 3.3 A System in which Machine Failures Require Reallocation

In many research efforts (e.g., [14], [17], [20]), the *flexibility* of a resource allocation has been closely tied to its robustness, and is described as the quality of the resource allocation that can allow it to be changed easily into another allocation of comparable performance when system failures occur. This section briefly sketches the use of the FePIA procedure to derive a robustness metric for systems where resource reallocation becomes necessary due to dynamic machine failures. In the example derivation analysis given below, it is assumed that resource reallocation is invoked because of permanent simultaneous failure of a number of machines in the system (e.g., due to a power failure in a section of a building).

For the system to be robust, it is required that 1) the total number of applications that need to be reassigned, $\underline{N^{\text{re-asgn}}}$, has to be less than $\underline{\tau_1}$ percent of the total number of applications and 2) the value of a given objective function (e.g., average application response time), $\underline{J}$, should not be any more than $\underline{\tau_2}$ times its value, $\underline{J^{\text{orig}}}$, for the original resource allocation. It is assumed that there is a specific resource reallocation algorithm which may not be the same as the original resource allocation algorithm. The resource reallocation algorithm will reassign the applications originally allocated to the failed machines to other machines, as well as reassign some other applications if necessary. As in Section 3.1, $\mathcal{A}$ and $\mathcal{M}$ are the sets of applications and machines, respectively.

Following Step 1 of the FePIA procedure, $\Phi = \{N^{\text{re-asgn}}, J\}$. Step 2 requires that the perturbation parameter $\boldsymbol{\pi}_j$ be identified. Let $\underline{\boldsymbol{F}}$ be a vector that indicates the identities of the machines that have failed. Specifically, $\boldsymbol{F} = [f_1\ f_2\ \cdots\ f_{|\mathcal{M}|}]^{\text{T}}$ such that $\underline{f_j}$ is 1 if $m_j$ fails, and is 0 otherwise. The vector $\boldsymbol{F}^{\text{orig}}$ corresponds to the original value of $\boldsymbol{F}$, which is $[0\ \ 0\ \ \cdots\ \ 0]^T$.

Step 3 asks for identifying the impact of $\boldsymbol{F}$ on $N^{\text{re-asgn}}$ and $J$. The impact depends on the resource reallocation algorithm, as well as $\boldsymbol{F}$, and can be determined from the resource allocation produced by the resource reallocation algorithm. Then, $N^{\text{re-asgn}}$ and $J$ can be reexpressed as functions of $\boldsymbol{F}$ as $N^{\text{re-asgn}}(\boldsymbol{F})$ and $J(\boldsymbol{F})$, respectively.

Following Step 4, the set of boundary values of $\boldsymbol{F}$ needs to be identified. However, $\boldsymbol{F}$ is a discrete variable. The boundary relationships developed for a continuous $\boldsymbol{\pi}_j$, i.e., $f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min}$ and $f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}$, will not apply because it is possible that no value of $\boldsymbol{\pi}_j$ will lie on the boundaries $\beta_i^{\min}$ and $\beta_i^{\max}$. Therefore, one needs to determine all those pairs of the values of $\boldsymbol{F}$ such that the values in a given pair bracket a given boundary ($\beta_i^{\min}$ or $\beta_i^{\max}$). For a given pair, the "boundary value" is taken to be the value that falls in the robust region. Let $\boldsymbol{F}^{(+1)}$ be a perturbation parameter value such that the machines that fail in the scenario represented by $\boldsymbol{F}^{(+1)}$ include the machines that fail in the scenario represented by $\boldsymbol{F}$ *and exactly* one other machine. Then, for $\phi_1 = N^{\text{re-asgn}}$, the set of "boundary values" for $\boldsymbol{F}$ is the set of all those "inner bracket" values of $\boldsymbol{F}$ for which the number of applications that need to be reassigned is less than the maximum tolerable number. Mathematically,

$$\Big\{\boldsymbol{F}|(N^{\text{re-asgn}}(\boldsymbol{F}) \le \tau_1|\mathcal{A}|) \wedge \Big(\exists \boldsymbol{F}^{(+1)}\ \ N^{\text{re-asgn}}(\boldsymbol{F}^{(+1)}) > \tau_1|\mathcal{A}|\Big)\Big\}.$$

For $\phi_2 = J$, the set of "boundary values" for $\boldsymbol{F}$ can be written as

$$\Big\{\boldsymbol{F}|\big(J(\boldsymbol{F}) \le \tau_2 J^{\text{orig}}\big) \wedge \Big(\exists \boldsymbol{F}^{(+1)}\quad J(\boldsymbol{F}^{(+1)}) > \tau_2 J^{\text{orig}}\Big)\Big\}.$$

Then, using (1), one can find the robustness radii for the set of constraints given above. However, for this system, it is more intuitive to use the $\underline{\ell_1\text{-norm}}$ (defined as $\sum_{r=1}^{n} |x_r|$ for a vector $\mathbf{x} = [x_1\ x_2\ \cdots\ x_n]^{\text{T}}$) for use in the robustness metric. This is because, with the $\ell_2$-norm, the term $\|\boldsymbol{F} - \boldsymbol{F}^{\text{orig}}\|_2$ equals the square root of the number of the machines that fail, rather than the (more natural) number of machines that fail. Specifically, using the $\ell_1$-norm,

$$r_\mu(N^{\text{re-asgn}}, \boldsymbol{F}) = \min_{\boldsymbol{F}:\ (N^{\text{re-asgn}}(\boldsymbol{F})] \le \tau_1|\mathcal{A}|) \wedge (\exists \boldsymbol{F}^{(+1)}\ \ N^{\text{re-asgn}}(\boldsymbol{F}^{(+1)}) > \tau_1|\mathcal{A}|)} \|\boldsymbol{F} - \boldsymbol{F}^{\text{orig}}\|_1 \quad (11)$$

and

$$r_\mu(J, \boldsymbol{F}) = \min_{\boldsymbol{F}:\ (J(\boldsymbol{F})] \le \tau_2 J^{\text{orig}}) \wedge (\exists \boldsymbol{F}^{(+1)}\ \ J(\boldsymbol{F}^{(+1)}) > \tau_2 J^{\text{orig}})} \|\boldsymbol{F} - \boldsymbol{F}^{\text{orig}}\|_1. \quad (12)$$

The robustness radius in (11) is the largest number of machines that can fail in any combination without causing the number of applications that have to be reassigned to exceed $\tau_1|\mathcal{A}|$. Similarly, the robustness radius in (12) is the largest number of machines that can fail in any combination without causing the objective function in the reallocated system to degrade beyond

$\tau_2 J(F)$. The robustness metric, from (2), is given by $\rho_\mu(\Phi, F) = \min(r_\mu(N^{\text{re-asgn}}, F),\ r_\mu(J, F)$. As in Section 3.2, it is assumed here that the discrete optimization problems posed in (11) and (12) can be solved for optimal or near-optimal solutions using combinatorial optimization techniques [19].

To determine if the robustness metric value is $k$, the reallocation algorithm must be run for all combinations of $k$ machines failures out of a total of $|\mathcal{M}|$ machines. Assuming that the robustness value is small enough, for example, five machine failures in a set of 100 machines, then the number of combinations would be small enough to be computed off-line in a reasonable time. If one is using a fast greedy heuristic for reallocation (e.g., those presented in [1]), the complexity would be $O(|\mathcal{A}||\mathcal{M}|)$ for each combination of failures considered. For a Min-min-like greedy heuristic (shown to be effective for many heterogeneous computing systems, see [1] and the references provided at the end of [1]), the complexity would be $O(|\mathcal{A}|^2|\mathcal{M}|)$ for each combination of failures considered.

## 4 Robustness against Multiple Perturbation Parameters

Section 2 developed the analysis for determining the robustness metric for a system with a single perturbation parameter. In this section, that analysis is extended to include multiple perturbation parameters.

Multiple perturbation parameters are considered by concatenating them into one parameter, which is then used as a single parameter as discussed in Section 2. Specifically, this section develops an expression for the robustness radius for a single performance feature, $\phi_i$, and multiple perturbation parameters. Then, the robustness metric is determined by taking the minimum over the robustness radii of all $\phi_i \in \Phi$.

Let the vector $\pi_j$ have $\underline{n_{\pi_j}}$ elements, and let $\star$ be the vector concatenation operator, $\underline{\text{so}}$ that $\pi_1 \star \pi_2 = [\pi_{11}\pi_{12} \ \cdots \pi_{1n_{\pi_1}} \ \pi_{21} \ \pi_{22} \ \cdots \ \pi_{2n_{\pi_2}}]^{\text{T}}$. Let $\underline{\mathbf{P}}$ be a weighted concatenation of the vectors $\pi_1, \pi_2, \cdots, \pi_{|\Pi|}$. That is, $\mathbf{P} = (\alpha_1 \times \pi_1) \star (\alpha_2 \times \pi_2) \star \cdots \star (\alpha_{|\Pi|} \times \pi_{|\Pi|})$, where $\underline{\alpha_j}$ $(1 \leq j \leq |\Pi|)$ is a weighting constant that may be assigned by a system administrator or be based on the sensitivity of the system performance feature $\phi_i$ toward $\pi_j$ (explained in detail later).

The vector $\mathbf{P}$ is analogous to the vector $\pi_j$ discussed in Section 2. Parallel to the discussion in Section 2, one needs to identify the set of boundary values of $\mathbf{P}$. Let $\underline{f_i}$ be a function that maps $\mathbf{P}$ to $\phi_i$. (Note that $f_i$ could be independent of some $\pi_j$.) For the single system feature $\phi_i$ being considered, such a set is given by $\{\mathbf{P}|(f_i(\mathbf{P}) = \beta_i^{\min}) \bigvee (f_i(\mathbf{P}) = \beta_i^{\max})\}$.

Let $\underline{\mathbf{P}^{\text{orig}}}$ be the assumed value of $\mathbf{P}$. In addition, let $\underline{\mathbf{P}^\star(\phi_i)}$ be analogous to $\pi_j^*(\phi_i)$, the element in the set of boundary values such that the Euclidean distance from $\mathbf{P}^{\text{orig}}$ to $\mathbf{P}^\star(\phi_i)$, $||\mathbf{P}^\star(\phi_i) - \mathbf{P}^{\text{orig}}||_2$, is the smallest over all such distances from $\mathbf{P}^{\text{orig}}$ to a point in the boundary set. Alternatively, the value $||\mathbf{P}^\star(\phi_i) - \mathbf{P}^{\text{orig}}||_2$ gives the largest Euclidean distance that the variable $\mathbf{P}$ can move in *any* direction from an assumed value of $\mathbf{P}^{\text{orig}}$ without exceeding the tolerable limits on $\phi_i$. Parallel to the discussion in Section 2, let the distance $||\mathbf{P}^\star(\phi_i) - \mathbf{P}^{\text{orig}}||_2$ be called the robustness radius, $r_\mu(\phi_i, \mathbf{P})$, of $\phi_i$ against $\mathbf{P}$. Mathematically,

$$r_\mu(\phi_i, \mathbf{P}) = \min_{\mathbf{P}:\ (f_i(\mathbf{P})=\beta_i^{\min}) \bigvee (f_i(\mathbf{P})=\beta_i^{\max})} ||\mathbf{P} - \mathbf{P}^{\text{orig}}||_2. \qquad (13)$$

Extending for all $\phi_i \in \Phi$, the robustness of resource allocation $\mu$ with respect to the performance feature set $\Phi$ against the perturbation parameter set $\Pi$ is given by $\rho_\mu(\Phi, \mathbf{P}) = \min_{\phi_i \in \Phi}(r_\mu(\phi_i, \mathbf{P}))$.

The sensitivity-based weighting procedure for the calculation of $\alpha_j$s is now discussed. Typically, $\pi_1, \pi_2, \cdots, \pi_{|\Pi|}$ will have different dimensions, i.e., will be measured in different units, e.g., seconds, objects per data set, bytes, etc. Before the concatenation of these vectors into $\mathbf{P}$, they should be converted into a single dimension. Additionally, for a given $\phi_i$, the magnitudes of $\alpha_j$ should indicate the relative sensitivities of $\phi_i$ to different $\pi_j$s. One way to accomplish the above goals is to set $\alpha_j = 1/r_\mu(\phi_i, \pi_j)$. With this definition of $\alpha_j$,

$$\mathbf{P} = \frac{\pi_1}{r_\mu(\phi_i, \pi_1)} \star \frac{\pi_2}{r_\mu(\phi_i, \pi_2)} \star \ \cdots \ \star \frac{\pi_{|\Pi|}}{r_\mu(\phi_i, \pi_{|\Pi|})}. \qquad (14)$$

Note that a smaller value of $r_\mu(\phi_i, \pi_j)$ makes $\alpha_j$ larger. This is desirable because a small value of the robustness against $\pi_j$ indicates that $\phi_i$ has a big sensitivity to changes in $\pi_j$ and, therefore, the relative weight of $\pi_j$ should be large. Also note that the units of $r_\mu(\phi_i, \pi_j)$ are the units of $\pi_j$. This fact renders $\mathbf{P}$ dimensionless.

## 5 Computational Complexity

To calculate the robustness radius, one needs to solve the optimization problem posed in (1). Such a computation could potentially be very expensive. However, one can exploit structure of this problem, along with some assumptions, to make this problem somewhat easier to solve. An optimization problem of the form $\min_{l(x)=0} f(x)$ or $\min_{c(x)\geq 0} f(x)$ could be solved very efficiently to find the global minimum if $f(x)$, $l(x)$, and $c(x)$ are convex, linear, and concave functions, respectively. Some solution approaches, including the well-known interior-point methods, for such *convex optimization* problems are presented in [5].

Because all norms are convex functions [5], the optimization problem posed in (1) reduces to a convex optimization problem if $f_{ij}(\pi_j)$ is linear. One interesting problem with linear $f_{ij}(\pi_j)$ is given in Section 3.1.

If $f_{ij}(\pi_j)$ is concave and the constraint "$f_{ij}(\pi_j) = \beta_i^{\min}$" is irrelevant for some scenario (as it is for the system in Section 3.2 where the latency of a path must be no larger than a certain limit, but can be arbitrarily small), then once again the problem reduces to a convex optimization problem. Because the distance from a point to the boundary of a region is the same as the distance from the point to the region itself, $\min_{\pi_j:\ (f_{ij}(\pi_j)=\beta_i^{\max})} ||\pi_j - \pi_j^{\text{orig}}||_2$ is equivalent to $\min_{\pi_j:\ (f_{ij}(\pi_j)\geq\beta_i^{\max})} ||\pi_j - \pi_j^{\text{orig}}||_2$. In such a case, the optimization problem would still be convex (and efficiently solvable) even if $f_{ij}(\pi_j)$ were concave [5].

Similarly, if $f_{ij}(\pi_j)$ is convex and the constraint "$f_{ij}(\pi_j) = \beta_i^{\max}$" is irrelevant for some scenario (e.g., for a

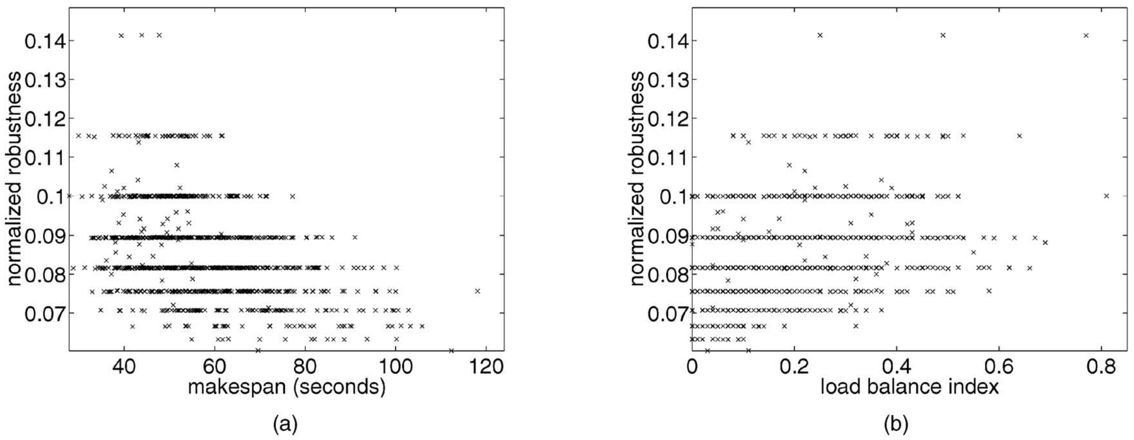


Fig. 3. The plots of normalized robustness against (a) makespan and (b) load balance index for 1,000 randomly generated resource allocations.

network, the throughput must be no smaller than a certain value, but can be arbitrarily large), then the optimization problem reduces to a convex optimization problem.

However, if the above conditions are not met, the optimization problem posed in (1) could still be solved for near-optimal solutions using heuristic approaches (some examples are given in [8]).

## 6 Experiments

The experiments in this section seek to establish the utility of the robustness metric in distinguishing between resource allocations that perform similarly in terms of a commonly used metric, such as makespan. Two different systems were considered: the independent task allocation system discussed in Section 3.1 and the HiPer-D system outlined in Section 3.2. Experiments were performed for a system with five machines and 20 applications. A total of 1,000 resource allocations were generated by assigning a randomly chosen machine to each application and then each resource allocation was evaluated with the robustness metric and the commonly used metric.

### 6.1 Independent Application Allocation System

For the system in Section 3.1, the ETC values were generated by sampling a Gamma distribution. The mean was arbitrarily set to 10, the *task heterogeneity* was set to 0.7, and the *machine heterogeneity* was also set to 0.7 (the heterogeneity of a set of numbers is the standard deviation divided by the mean). See [2] for a description of a method for generating random numbers with given mean and heterogeneity values.

The resource allocations were evaluated for robustness, makespan, and *load balance index* (defined as the ratio of the finishing time of the machine that finishes first to the makespan). The larger the value of the load balance index, the more balanced the load (the largest value being 1). The tolerance, $\tau$, was set to 120 percent (i.e., the actual makespan could be no more than 1.2 times the nominal value). In this context, a robustness value of $x$ for a given resource allocation means that the resource allocation can endure any combination of ETC errors without the makespan increasing beyond 1.2 times its nominal value as long as the Euclidean norm of the errors is no larger than $x$ seconds.

Fig. 3a shows the "normalized robustness" of a resource allocation against its makespan. The *normalized robustness* equals the absolute robustness divided by the predicted makespan. A similar graph for the normalized robustness against the load balance index is shown in Fig. 3b. It can be seen in Fig. 3 that some resource allocations are clustered into groups such that, for all resource allocations within a group, the normalized robustness remains constant as the predicted makespan (or load balance index) increases.

The cluster of the resource allocations with the highest robustness has the feature that the most loaded machine has the smallest number of applications allocated to it (which is two for the experiments in Fig. 3). The cluster with the smallest robustness has the largest number, 11, of applications allocated to it. The intuitive explanation for this behavior is that the larger the number of applications allocated to a machine, the more degrees of freedom for the finishing time of that machine. A larger degree of freedom then results in a shorter path to constraint violation in the parameter space. That is, the robustness is then smaller (using the $\ell_2$-norm).

If one agrees with the utility of the observations made above, one can still question if the same information could be gleaned from some traditional metrics (even if they are not traditionally used to measure robustness). In an attempt to answer that question, note that sharp differences exist in the robustness of some resource allocations that have very similar values of makespan. A similar observation could be made from the robustness against load balance index plot (Fig. 3b). In fact, it is possible to find a set of resource allocations that have very similar values of makespan and very similar values of load balance index, but with very different values of robustness. These observations highlight the fact that the information given by the robustness metric could not be obtained from two popular performance metrics.

The clustering seen in Fig. 3 can be explained using (5). Let $\underline{m(\boldsymbol{C})}$ be the machine that determines the makespan at $\boldsymbol{C}$. Let $n(m_j)$ be the number of applications allocated to machine $m_j$. If $m(\boldsymbol{C}^{\text{orig}})$ has the largest number of applications allocated to it, then it is also the machine that determines the robustness of the resource allocation (because it has the smallest robustness radius, (see (5)). Now, consider the set $S_1(x)$ of resource allocations such that $x = n(m(\boldsymbol{C}^{\text{orig}})) = \max_{j:m_j \in \mathcal{M}} n(m_j)$ for each resource allocation in the set. For resource allocations in $S_1(x)$, the robustness is directly proportional to $M^{\text{orig}}$ (see (5)) or, equivalently, the normalized robustness is a constant as $M^{\text{orig}}$ changes. Each distinct straight line in Fig. 3 corresponds to $S_1(x)$ for some $x \in \{1 \cdots |\mathcal{A}|\}$. The explanation for the outlying points is as follows: Let $S_2(x)$ be the union of $S_1(x)$ and the set of resource allocations for which $x = n(m(\boldsymbol{C}^{\text{orig}})) \neq \max_{j:m_j \in \mathcal{M}} n(m_j)$. The outlying points belong to the latter set, $S_2(x) - S_1(x)$. Note that all such outlying points lie "below" the line specified by $S_1(x)$. For a resource allocation that corresponds to an outlying point, the machine that determines the robustness is not $m(\boldsymbol{C}^{\text{orig}})$; it is some other machine for which the robustness radius is smaller than the robustness radius for $m(\boldsymbol{C}^{\text{orig}})$.

### 6.2 The HiPer-D System

For the model in Section 3.2, the experiments were performed for a system that consisted of 19 paths, where the end-to-end latency constraints of the paths were uniformly sampled from the range [750, 1250]. The system had three sensors (with rates $4 \times 10^{-5}$, $3 \times 10^{-5}$, and $8 \times 10^{-6}$) and three actuators. The experiments made the following simplifying assumptions. The computation time function, $T^c_{ij}(\boldsymbol{\lambda})$, was assumed to be of the form $\sum_{1 \le z \le 3} b_{ijz} \boldsymbol{\lambda}_z$, where $b_{ijz} = 0$ if there is no route from the $z$th sensor to application $a_i$. Otherwise, $b_{ijz}$ was sampled from a Gamma distribution with a mean of 10 and task and machine heterogeneity values of 0.7 each. For simplicity in the presentation of the results, the communication times were all set to zero. These assumptions were made only to simplify the experiments and are *not* a part of the formulation of the robustness metric. The salient point in this example is that the utility of the robustness metric can be seen even when simple complexity functions are used.

The resource allocations were evaluated for robustness and "slack." In this context, a robustness value of $x$ for a given resource allocation means that the resource allocation can endure any combination of sensor loads without a latency or throughput violation as long as the Euclidean norm of the increases in sensor loads (from the assumed values) is no larger than $x$. Slack has been used in many studies as a performance measure (e.g., [10], [17]) for resource allocation in parallel and distributed systems, where a resource allocation with a larger slack is considered better. In this study, slack is defined mathematically as follows: Let the *fractional value* of a given QoS attribute be the value of the attribute as a percentage of the maximum allowed value. Then, the *percentage slack* for a given QoS attribute is the fractional value subtracted from 1. The *system-wide percentage slack* is the minimum value of percentage slack taken over all QoS constraints, and can be expressed mathematically as

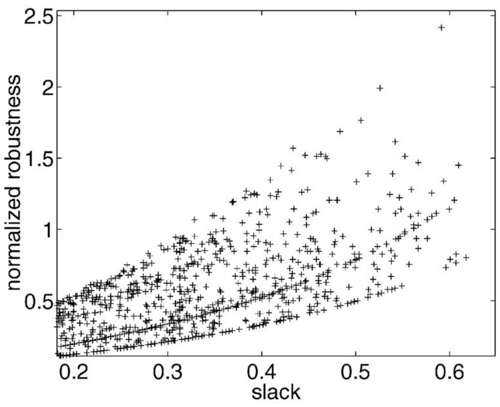


Fig. 4. The plot of normalized robustness against slack for 1,000 randomly generated resource allocations.

$$\min \left[ \min_{k:\, \mathcal{P}_k \in \mathcal{P}} \left( 1 - \frac{L_k(\boldsymbol{\lambda})}{L_k^{\max}} \right), \min_{i:\, a_i \in \mathcal{A}} \left( 1 - \frac{\max\left(T^c_{ij}(\boldsymbol{\lambda}),\ \max_{a_p \in \mathcal{D}(a_i)} T^n_{ip}(\boldsymbol{\lambda})\right)}{1/R(a_i)} \right) \right]. \tag{15}$$

Fig. 4 shows the normalized robustness of a resource allocation against its slack. For this system, the normalized robustness equals the absolute robustness divided by $||\boldsymbol{\lambda}^{\text{orig}}||_2$. It can be seen that the normalized robustness and slack are not correlated. ***If***, in some research study, the purpose of using slack is to measure a system's ability to tolerate additional load, ***then*** our measure of robustness is a better indicator of that ability than slack. This is because the expression for slack, (15), does not directly take into account how the sensor loads affect the computation and communication times. It could be conjectured that, for a system where all sensors affected the computation and communication times of all applications in exactly the same way, the slack and this research's measure of robustness would be tightly correlated. This, in fact, is true. Other experiments performed in this study show that, for a system with small heterogeneity, the robustness and slack are tightly correlated, thereby suggesting that robustness measurements are not needed if slack is known. As the system heterogeneity increases, the robustness and slack become less correlated, indicating that the robustness measurements can be used to distinguish between mappings that are similar in terms of the slack. As the system size increases, the correlation between the slack and the robustness decreases even further. In summary, for heterogeneous systems, using slack as a measure of how much increase in sensor load a system can tolerate may cause system designers to grossly misjudge the system's capability.

## 7 Related Work

Although a number of robustness measures have been studied in the literature (e.g., [4], [7], [9], [10], [11], [12], [17], [18], [20], [22]), those measures were developed for specific systems. The focus of the research in this paper is a general mathematical formulation of a robustness metric that could be applied to a variety of parallel and distributed systems by following the FePIA procedure presented in this paper.

Given an allocation of a set of communicating applications to a set of machines, the work in [4] develops a metric for the robustness of the makespan against uncertainties in the estimated execution times of the applications. The paper discusses, in detail, the effect of these uncertainties on the value of makespan and how the robustness metric could be used to find more robust resource allocations. Based on the model and assumptions in [4], several theorems about the properties of robustness are proven. The robustness metric in [4] was formulated for errors in the estimation of application execution times and was not intended for general use (in contrast to our work). Additionally, the formulation in [4] assumes that the execution time for any application is at most $k$ times the estimated value, where $k \geq 1$ is the same for all applications. In our work, no such bound is assumed.

In [7], the authors address the issue of probabilistic guarantees for fault-tolerant real-time systems. As a first step toward determining such a probabilistic guarantee, the authors determine the maximum frequency of software or hardware faults that the system can tolerate without violating any hard real-time constraint. In the second step, the authors derive a value for the probability that the system will not experience faults at a frequency larger than that determined in the first step. The output of the first step is what our work would identify as the robustness of the system, with the satisfaction of the real-time constraints being the robustness requirement and the occurrence of faults being the perturbation parameter.

The research in [9] considers a single-machine scheduling environment where the processing times of individual jobs are uncertain. The system performance is measured by the total flow time (i.e., the sum of *completion* times of all jobs). Given the probabilistic information about the processing time for each job, the authors determine the normal distribution that approximates the flow time associated with a given schedule. A given schedule's robustness is then given by 1 minus the risk of achieving substandard flow time performance. The risk value is calculated by using the approximate distribution of flow time.

The study in [10] explores slack-based techniques for producing robust resource allocations in a job-shop environment. The central idea is to provide each task with extra time (defined as slack) to execute so that some level of uncertainty can be absorbed without having to reallocate. The study uses slack as its measure of robustness.

The Ballista project [11] explores the robustness of commercial off-the-shelf software against failures resulting from invalid inputs to various software procedure calls. A failure causes the software package to crash when unexpected parameters are used for the procedure calls. The research quantifies the robustness of a software procedure in terms of its failure rate—the percentage of test input cases that cause failures to occur. The Ballista project extensively explores the robustness of different operating systems (including experimental work with IBM, FreeBSD, Linux, AT&T, and Cisco).

The research in [12] introduces techniques to incorporate fault tolerance in scheduling approaches for real-time systems by the use of additional time to perform the system functions (e.g., to reexecute, or to execute a different version of, a faulty task). Their method guarantees that the real-time tasks will meet the deadlines under transient faults by reserving sufficient additional time or slack. Given a certain system slack and task model, the paper defines its measure of robustness to be the "fault tolerance capability" of a system (i.e., the number and frequency of faults it can tolerate). This measure of robustness is similar, in principle, to ours.

In [17], a "neighborhood-based" measure of robustness is defined for a job-shop environment. Given a schedule $s$ and a performance metric $P(s)$, the robustness of the schedule $s$ is defined to be a weighted sum of all $P(s')$ values such that $s'$ is in the set of schedules that can be obtained from $s$ by interchanging two consecutive operations on the same machine.

The work in [18] develops a mathematical definition for the robustness of makespan against machine breakdowns in a job-shop environment. The authors assume a certain random distribution of the machine breakdowns and a certain rescheduling policy in the event of a breakdown. Given these assumptions, the robustness of a schedule $s$ is defined to be a weighted sum of the expected value of the makespan of the rescheduled system, $M$, and the expected value of the schedule delay (the difference between $M$ and the original value of the makespan). Because the analytical determination of the schedule delay becomes very hard when more than one disruption is considered, the authors propose surrogate measures of robustness that are claimed to be strongly correlated with the expected value of $M$ and the expected schedule delay.

The research in [20] uses a genetic algorithm to produce robust schedules in a job-shop environment. Given a schedule $s$ and a performance metric $P(s)$, the "robust fitness value" of the schedule $s$ is a weighted average of all $P(s')$ values such that $s'$ is in a set of schedules obtained from $s$ by adding a small "noise" to it. The size of this set of schedules is determined arbitrarily. The "noise" modifies $s$ by randomly changing the ready times of a fraction of the tasks.

Our work is perhaps closest in philosophy to [22], which attempts to calculate the stability radius of an optimal schedule in a job-shop environment. The stability radius of an optimal schedule, $s$, is defined to be the radius of a closed ball in the space of the numerical input data such that, within that ball, the schedule $s$ remains optimal. Outside this ball, which is centered at the assumed input, some other schedule would outperform the schedule that is optimal at the assumed input. From our viewpoint, for a given optimal schedule, the robustness requirement could be the persistence of optimality in the face of perturbations in the input data. Our work differs and is more general because we consider the given system requirements to generate a robustness requirement and then determine the robustness. In addition, our work considers the possibility of multiple perturbations in different dimensions.

## 8 Conclusions

This paper has presented a mathematical description of a new metric for the robustness of a resource allocation with respect to desired system performance features against multiple perturbations in various system and environmental conditions. In addition, the research describes a procedure, called FePIA, to methodically derive the robustness metric for a variety of parallel and distributed resource allocation systems. For illustration, the FePIA procedure is employed to derive robustness metrics for three example distributed systems. The experiments conducted in this research for two example parallel and distributed systems illustrate the utility of the robustness metric in distinguishing between the resource allocations that perform similarly otherwise.

## ACKNOWLEDGMENTS

The authors thank the *IEEE Transactions on Parallel and Distributed Systems* reviewers, Professor Balakrishnan, Sahra Sedigh-Ali, and Martin Nagel for their valuable comments which improved the quality of this paper. A preliminary version of portions of this paper was presented at the 17th International Parallel and Distributed Processing Symposium, April 2003. This work was supported by the DARPA/ITO Quorum Program through the Office of Naval Research under Grant No. N00014-00-1-0599, and by the Colorado State University George T. Abell Endowment. Some of the equipment used was donated by Intel and Microsoft.

## REFERENCES

[1] S. Ali, J.-K. Kim, Y. Yu, S.B. Gundala, S. Gertphol, H.J. Siegel, A.A. Maciejewski, and V. Prasanna, "Greedy Heuristics for Resource Allocation in Dynamic Distributed Real-Time Heterogeneous Computing Systems," *Proc. 2002 Int'l Conf. Parallel and Distributed Processing Techniques and Applications,* vol. 2, pp. 519-530, June 2002.

[2] S. Ali, H.J. Siegel, M. Maheswaran, D. Hensgen, and S. Sedigh-Ali, "Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems," *Tamkang J. Science and Eng.,* vol. 3, no. 3, pp. 195-207, Nov. 2000.

[3] P.M. Berry, "Uncertainty in Scheduling: Probability, Problem Reduction, Abstractions and the User," *IEE Computing and Control Division Colloquium on Advanced Software Technologies for Scheduling, Digest No: 1993/163,* Apr. 1993.

[4] L. Bölöni and D.C. Marinescu, "Robust Scheduling of Metaprograms," *J. Scheduling,* vol. 5, no. 5, pp. 395-412, Sept. 2002.

[5] S. Boyd and L. Vandenberghe, "Convex Optimization," http://www.stanford.edu/class/ee364/index.html, 2003.

[6] T.D. Braun, H.J. Siegel, N. Beck, L.L. Bölöni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, B. Yao, D. Hensgen, and R.F. Freund, "A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems," *J. Parallel and Distributed Computing,* vol. 61, no. 6, pp. 810-837, June 2001.

[7] A. Burns, S. Punnekkat, B. Littlewood, and D. Wright, "Probabilistic Guarantees for Fault-Tolerant Real-Time Systems," Technical Report, Design for Validation (DeVa) TR No. 44, Esprit Long Term Research Project No. 20072, Dept. of Computer Science, Univ. of Newcastle upon Tyne, U.K., 1997.

[8] Y.X. Chen, "Optimal Anytime Search for Constrained Nonlinear Programming," master's thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, May 2001.

[9] R.L. Daniels and J.E. Carrillo, "$\beta$-Robust Scheduling for Single-Machine Systems with Uncertain Processing Times," *IIE Trans.,* vol. 29, no. 11, pp. 977-985, 1997.

[10] A.J. Davenport, C. Gefflot, and J.C. Beck, "Slack-Based Techniques for Robust Schedules," *Proc. Sixth European Conf. Planning,* pp. 7-18, Sept. 2001.

[11] J. DeVale and P. Koopman, "Robust Software—No More Excuses," *Proc. IEEE Int'l Conf. Dependable Systems and Networks,* pp. 145-154, June 2002.

[12] S. Ghosh, "Guaranteeing Fault Tolerance through Scheduling in Real-Time Systems," PhD thesis, Faculty of Arts and Sciences, Univ. of Pittsburgh, 1996.

[13] S.D. Gribble, "Robustness in Complex Systems," *Proc. Eighth Workshop Hot Topics in Operating Systems,* pp. 21-26, May 2001.

[14] E. Hart, P.M. Ross, and J. Nelson, "Producing Robust Schedules via an Artificial Immune System," *Proc. 1998 Int'l Conf. Evolutionary Computing,* pp. 464-469, May 1998.

[15] R. Harrison, L. Zitzman, and G. Yoritomo, "High Performance Distributed Computing Program (HiPer-D)—Engineering Testbed One (T1) Report," technical report, Naval Surface Warfare Center, Dahlgren, Va., Nov. 1995.

[16] E. Jen, "Stable or Robust? What is the Difference?" *Complexity,* to appear.

[17] M. Jensen, "Improving Robustness and Flexibility of Tardiness and Total Flowtime Job Shops Using Robustness Measures," *J. Applied Soft Computing,* vol. 1, no. 1, pp. 35-52, June 2001.

[18] V.J. Leon, S.D. Wu, and R.H. Storer, "Robustness Measures and Robust Scheduling for Job Shops," *IEE Trans.,* vol. 26, no. 5, pp. 32-43, Sept. 1994.

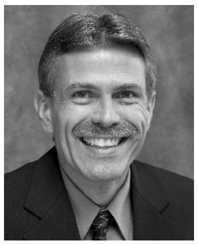
[19] G.L. Nemhauser and L.A. Wolsey, *Integer and Combinatorial Optimization.* New York: John Wiley & Sons, 1988.

[20] M. Sevaux and K. Sörensen, "Genetic Algorithm for Robust Schedules," *Proc. Eighth Int'l Workshop Project Management and Scheduling,* pp. 330-333, Apr. 2002.

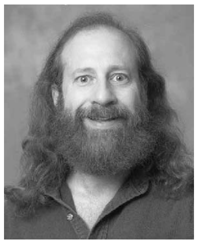
[21] G.F. Simmons, *Calculus With Analytic Geometry,* second ed. New York: McGraw-Hill, 1995.

[22] Y.N. Sotskov, V.S. Tanaev, and F. Werner, "Stability Radius of an Optimal Schedule: A Survey and Recent Developments," *Industrial Applications of Combinatorial Optimization,* G. Yu, ed., Norwell, Mass.: Kluwer Academic Publishers, pp. 72-108, 1998.

**Shoukat Ali** received the MSEE (1999) and PhD (2003) degrees from the School of Electrical and Computer Engineering at Purdue University, West Lafayette. He received the BS degree (1996) in electrical engineering from the University of Engineering and Technology (UET), Lahore, Pakistan. He is an assistant professor in the Department of Electrical and Computer Engineering at the University of Missouri-Rolla. His research interests include heterogeneous parallel and distributed computing and communication systems. He has coauthored 12 published technical papers in this area. He was the publicity cochair of the 11th IEEE Heterogeneous Computing Workshop (2002), and is on the program committee for the 12th IEEE Heterogeneous Computing Workshop (2003). He is a member of the IEEE, IEEE Computer Society, and ACM.



**Anthony A. Maciejewski** received the BSEE, MS, and PhD degrees in electrical engineering in 1982, 1984, and 1987, respectively, all from The Ohio State University. From 1985 to 1986, he was an American Electronics Association Japan Research Fellow at the Hitachi Central Research Laboratory in Tokyo, Japan. From 1988 to 2001, he was a professor of electrical and computer engineering at Purdue University, West Lafayette. In 2001, he joined Colorado State University where he is currently the head of the Department of Electrical and Computer Engineering. Professor Maciejewski's research interests are in robotics and high-performance computing. He is currently an associate editor for the *IEEE Transactions on Robotics and Automation*, serves on the Administrative Committee for the IEEE Robotics and Automation Society, and was the technical program chair for the 2002 IEEE International Conference on Robotics and Automation. An up-to-date biography is available at http://www.engr.colostate.edu/~aam. He is a senior member of the IEEE, IEEE Computer Society, and ACM.



**Howard Jay Siegel** received two BS degrees from the Massachusetts Institute of Technology (MIT), and the MA, MSE, and PhD degrees from Princeton University. He was appointed the George T. Abell Endowed Chair Distinguished Professor of electrical and computer engineering at Colorado State University (CSU) in August 2001, where he is also a professor of computer science. In December 2002, he became the first director of the university-wide CSU Information Science and Technology Center (ISTeC). From 1976 to 2001, he was a professor at Purdue University. He has coauthored more than 300 published papers on parallel and distributed computing and communication, is a fellow of the IEEE and IEEE Computer Society, a fellow of the ACM, was a coeditor-in-chief of the *Journal of Parallel and Distributed Computing*, and was on the editorial boards of both the *IEEE Transactions on Parallel and Distributed Systems* and the *IEEE Transactions on Computers*. He was program chair/cochair of three major international conferences, general chair/cochair of four international conferences, and chair/cochair of five workshops. He has been an international keynote speaker and tutorial lecturer, and has consulted for industry and government.

**Jong-Kook Kim** received the MS degree in electrical engineering from Purdue University in May 2000. He received the BS degree in electronic engineering from Korea University, Seoul, Korea in 1998. He is pursuing the PhD degree from the School of Electrical and Computer Engineering at Purdue University, where he is currently a research assistant (since August 1998). He has presented his work at several international conferences and has been a reviewer for numerous conferences and journals. His research interests include heterogeneous distributed computing, computer architecture, performance measure, resource management, evolutionary heuristics, and power-aware computing. He is a student member of the IEEE, IEEE Computer Society, and ACM.