Measuring the Robustness of a Resource Allocation

Shoukat Ali[†], Anthony A. Maciejewski[‡], Howard Jay Siegel[‡§], and Jong-Kook Kim[†]

[†]Purdue University
School of Electrical and
Computer Engineering
West Lafayette, IN 47907-1285 USA
{alis, jongkook}@purdue.edu

Colorado State University
[‡]Department of Electrical and
Computer Engineering
[§]Department of Computer Science
Fort Collins, CO 80523-1373 USA
{hj, aam}@colostate.edu

January 2003

Submitted to the IEEE Transactions on Parallel and Distributed Systems
Corresponding author: Shoukat Ali

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.

Keywords: robustness, robustness metric, resource allocation, resource management systems, parallel and distributed systems.

This research 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.

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, 7, 11, 12, 15, 16, 17, 18, 19, 22]). 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 [18], 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, [12] states that a robust system continues to operate correctly across a wide range of operational conditions. Robustness, according to [7] and [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 [7] and [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 mapping (matching of applications to resources and ordering their execution (e.g., [6, 20, 24])) so as to maximize robustness of desired system features against inaccuracies in estimated system parameters and changes in the environment. This research addresses the design of a robustness metric for mappings.

A mapping 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 mapping 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 mapping, 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 mapping with respect to certain system performance features against multiple perturbations in multiple system components and environmental conditions. In addition, this work outlines a four-step procedure for deriving a robustness metric for an arbitrary system. For illustration, the four-step procedure is employed to derive robustness metrics for three example distributed systems. The robustness metric and the four-step procedure for its derivation are the main contributions of this paper.

The rest of the paper is organized as follows. A sampling of the related work is given in Section 2. Section 3 describes the four-step derivation 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 4. Section 5 extends the definition of the robustness metric given in Section 3 to multiple specified perturbation parameters. Section 6 presents some experiments that highlight the usefulness of the robustness metric. Section 7 concludes the paper.

2 Related Work

Although a number of robustness measures have been studied in the literature (e.g., [4, 8, 9, 13, 18, 19, 22]), those measures were developed for specific systems. Unlike those efforts, this paper presents a general mathematical formulation of a robustness metric that could be applied to a variety of parallel and distributed systems by following the four-step derivation procedure presented in this paper.

Given a mapping of a set of communicating applications to a set of machines, the work in [4] develops a metric for the robustness of the makespan[1] 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 mappings. 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 a specific bound on the only perturbation parameter considered in the study. In particular, it is assumed 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.

The study in [8] explores slack-based techniques for producing robust mappings 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 re-map. The study does not develop a robustness metric; instead, it implicitly uses slack to achieve robustness.

The Ballista project [9] 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. However, the robustness

[1]Makespan of a set of applications is the completion time for the entire set.

metric developed for that project is specific to software systems.

The research in [13] investigates an artificial immune system approach to producing robust schedules for a dynamic jobshop scheduling problem. The study considers a pair of schedules to be robust if there is a large similarity (based on a similarity measure defined in the paper) between the two schedules, indicating that a switch from one schedule to another could be performed with minimal disruption if rescheduling is required. Although the paper defines a measure of similarity between two mappings, it does not develop a robustness measure for a given mapping nor does it state the parameters against which the robustness is being sought.

In [18], 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. While [18] clearly defines a robustness measure for a given mapping, it does so only for the specific job-shop environment. Like [13], [18] does not explicitly state the perturbations towards which the system is robust.

The work in [19] 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 [22] 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. Like [13] and [18], [22] does not explicitly state the perturbations under which the system is robust.

All the measures of robustness developed in the research efforts discussed above are applicable only to specific systems. However, our work presents a general formulation of a robustness metric that can be applied to a variety of parallel and distributed systems, and can be used when there are multiple perturbation parameters.

3 Generalized Robustness Metric

The key contribution of the work presented here is the design of a general mathematical methodology for deriving the range of uncertainty in system parameters within which a desired level of quality of service ($\underline{\text{QoS}}$) can be guaranteed. Central to achieving this goal is the development of a general, mathematically precise, definition of robustness that can be applied in a wide variety of scenarios, for a wide variety of application-specific measures of performance, and a wide variety of system parameters, whose behavior is uncertain, but whose values will affect system performance. This paper proposes a general four-step procedure for deriving such a robustness metric for any desired computing environment. Informally, the steps are:

I. Identify the QoS performance features that are of importance and 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 (assignment of machines to applications),

(b) the acceptable variation is up to 130% 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.

II. Identify all of the system and environment parameters whose values may impact the QoS performance features selected in step I. 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% of its estimated value) with respect to uncertainties in these estimated execution times.

III. Determine the exact impact each perturbation parameter has on each performance feature. 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 I(b) implies that the actual time each machine finishes its applications must be within the acceptable variation.

IV. The last step is to determine the smallest collective variation in the values of perturbation parameters identified in step II that will cause any of the performance features identified in step I 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% of its estimated value.

The above four-step procedure for deriving a robustness metric is referred to in this paper as the FePIA procedure, where the abbreviation stands for identifying the performance features, the perturbation parameters, the impact of perturbation parameters on performance features, and the analysis 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 more formally.

1. Describe quantitatively the requirement that makes the system robust. Based on this *robustness requirement*, determine the system performance features that should be limited in variation to ensure that the robustness requirement is met. Mathematically, let $\underline{\Phi}$ be the set of such system features. For each element $\phi_i \in \Phi$, quantitatively describe the tolerable variation in ϕ_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 ϕ_i. For the makespan example, ϕ_i is the time the i-th machine finishes its assigned applications, and its corresponding $\left\langle \beta_i^{\min}, \beta_i^{\max} \right\rangle$ could be $\langle 0,\ 1.3 \times (\text{estimated makespan value}) \rangle$.

2. Determine the system and environment perturbation parameters against which the robustness of the system features described in item 1 is sought. Mathematically, let $\underline{\Pi}$ be the set of such system and environment parameters. It is assumed that the elements of Π are vectors. Let $\boldsymbol{\pi}_j$ be the j-th element of Π. For the makespan example, $\boldsymbol{\pi}_j$ could be the vector composed of the actual application execution times, i.e., the i-th element of $\boldsymbol{\pi}_j$ is the actual execution time of the i-th application on the machine it was assigned.

3. Identify the impact of the perturbation parameters in item 2 on the system performance features in item 1. Mathematically, for every $\phi_i \in \Phi$, determine the relationship $\phi_i = f_{ij}(\boldsymbol{\pi}_j)$, if any, that relates ϕ_i to $\boldsymbol{\pi}_j$. In this expression, f_{ij} is a function that maps $\boldsymbol{\pi}_j$ to ϕ_i. For the makespan example, ϕ_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. Note that this expression assumes that each $\boldsymbol{\pi}_j \in \Pi$ affects a given ϕ_i independently. The case where multiple perturbation parameters can affect a given ϕ_i simultaneously will be examined in Section 5. The rest of this discussion will be developed assuming only one element in Π.

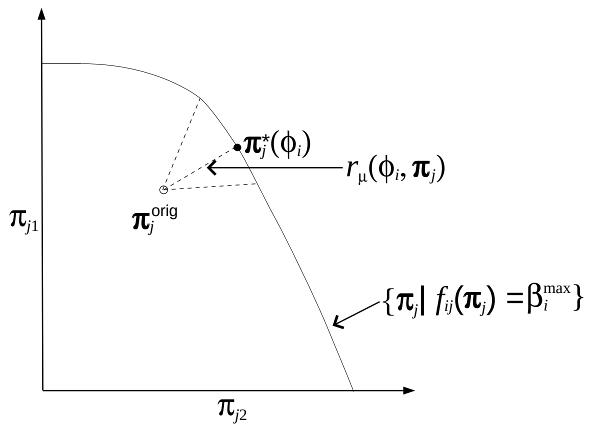
4. For every $\phi_i \in \Phi$, determine the *boundary values of* $\boldsymbol{\pi}_j$, i.e., the values satisfying the boundary relationships $f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min}$ and $f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}$. (If $\boldsymbol{\pi}_j$ is a discrete variable then the boundary values correspond to the closest values that bracket each boundary relationship. See Subsection 4.3 for an example.) These relationships separate the region of robust operation from that of non-robust operation. Find the smallest perturbation in $\boldsymbol{\pi}_j$ that causes any $\phi_i \in \Phi$ to exceed the bounds $\left\langle \beta_i^{\min}, \beta_i^{\max} \right\rangle$ 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 and/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. Figure 1 illustrates this concept for a single feature, ϕ_i, and a 2-element perturbation vector $\boldsymbol{\pi}_j \in \mathbf{R}^2$. The curve shown in Figure 1 plots the set of boundary points $\{\boldsymbol{\pi}_j |\ f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}\}$ for a mapping $\underline{\mu}$. For this figure, the set of boundary points $\left\{\boldsymbol{\pi}_j |\ f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min}\right\}$ is given by the points on the π_{j1}-axis and π_{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 ϕ_i. For a vector $\mathbf{x} = [x_1\, x_2\, \cdots\, x_n]^{\mathrm{T}}$, let $\underline{\|\mathbf{x}\|_2}$ be the ℓ_2-norm (Euclidean norm) of the vector, and be defined by $\sqrt{\sum_{r=1}^{n} x_r^2}$. The point on the curve marked as $\underline{\boldsymbol{\pi}_j^{\star}(\phi_i)}$ has the feature that the Euclidean distance from $\boldsymbol{\pi}_j^{\text{orig}}$ to $\boldsymbol{\pi}_j^{\star}(\phi_i)$, $\|\boldsymbol{\pi}_j^{\star}(\phi_i) - \boldsymbol{\pi}_j^{\text{orig}}\|_2$, is the smallest over all such distances from $\boldsymbol{\pi}_j^{\text{orig}}$ to a point on the curve. An important interpretation of $\boldsymbol{\pi}_j^{\star}(\phi_i)$ is that the value $\|\boldsymbol{\pi}_j^{\star}(\phi_i) - \boldsymbol{\pi}_j^{\text{orig}}\|_2$ 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 ϕ_i exceeding the tolerable variation. Let the distance $\|\boldsymbol{\pi}_j^{\star}(\phi_i) - \boldsymbol{\pi}_j^{\text{orig}}\|_2$ be called the $\underline{\text{robustness}}$ $\underline{\text{radius}}$, $\underline{r_\mu(\phi_i,\ \boldsymbol{\pi}_j)}$, of ϕ_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 metric for the robustness of mapping μ with respect to performance feature ϕ_i against the perturbation parameter $\boldsymbol{\pi}_j$.



$\boldsymbol{\pi}_j^{\star}(\phi_i)$
$r_\mu(\phi_i, \boldsymbol{\pi}_j)$
π_{j1}
$\boldsymbol{\pi}_j^{\text{orig}}$
$\{\boldsymbol{\pi}_j \mid f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}\}$
π_{j2}

Figure 1: The 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, $\{\boldsymbol{\pi}_j \mid f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}\}$. The set of boundary points, $\{\boldsymbol{\pi}_j \mid f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min}\}$ is given by the points on the π_{j1}-axis and π_{j2}-axis.

The metric definition can be extended easily for all $\phi_i \in \Phi$. It is simply the minimum of all robustness radii. Mathematically, let $\underline{\rho_\mu(\Phi,\ \boldsymbol{\pi}_j)}$ be the *robustness of mapping μ with respect to the performance feature set Φ against the perturbation parameter $\boldsymbol{\pi}_j$*. Then,

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

4 Derivations of Robustness Metric for Example Systems

4.1 Independent Application Allocation System

The first example derivation of the robustness metric is for a system that maps a set of independent applications on a set of machines [6]. In this system, it is required that makespan be robust against errors in application execution time estimates.

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 mapped on 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 multi-tasking), 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 mapping μ 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 mapped on it.

Assume that unknown inaccuracies in the ETC values are expected, requiring that the mapping μ be robust against them. More specifically, it is required that, for a given mapping, its actual makespan value $\underline{M}$ (calculated considering the effects of ETC errors) may be no more than τ 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 3, the system performance features that should be limited in variation to ensure the makespan robustness are the finishing times of the machines. That is,

$$\Phi = \{F_j \mid 1 \leq j \leq |\mathcal{M}|\}. \tag{3}$$

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 mapped.

Let $\underline{C_i}$ be equal to the actual computation time value (C_i^{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, $\underline{\boldsymbol{C}^{\text{orig}}} = [C_1^{\text{orig}} \;\; C_2^{\text{orig}} \;\; \cdots \;\; C_{|\mathcal{A}|}^{\text{orig}}]$. 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,

$$F_j(\boldsymbol{C}) = \sum_{i:\, a_i \text{ is mapped to } m_j} C_i. \tag{4}$$

Note that the finishing time of a given machine depends only on the actual execution times of the applications mapped 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 Equation 3 is given by $\left\{F_j(\boldsymbol{C}) = \tau M^{\text{orig}} \middle| \; 1 \le j \le |\mathcal{M}|\right\}$.

For a two-application system, $\boldsymbol{C}$ corresponds to $\boldsymbol{\pi}_j$ in Figure 1. Similarly, C_1 and C_2 correspond to π_{j1} and π_{j2}, respectively. The terms $\boldsymbol{C}^{\text{orig}}$, $F_j(\boldsymbol{C})$, and τM^{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 Equation 1, the robustness radius of F_j against $\boldsymbol{C}$ is given by

$$r_\mu(F_j,\ \boldsymbol{C}) = \min_{\boldsymbol{C}:\ F_j(\boldsymbol{C}) = \tau M^{\text{orig}}} \|\boldsymbol{C} - \boldsymbol{C}^{\text{orig}}\|_2 \tag{5}$$

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 τ times the nominal makespan value.

Note that the right hand side in Equation 5 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 [23], Equation 5 reduces to

$$r_\mu(F_j,\ \boldsymbol{C}) = \frac{\tau M^{\text{orig}} - F_j(\boldsymbol{C}^{\text{orig}})}{\sqrt{\text{number of applications mapped to } m_j}} \tag{6}$$

The robustness metric, from Equation 2, is

$$\rho_\mu(\Phi,\ \boldsymbol{C}) = \min_{F_j \in \Phi} r_\mu(F_j,\ \boldsymbol{C}) = \min_{F_j \in \Phi} \left(\min_{\boldsymbol{C}:\ F_j(\boldsymbol{C}) = \tau M^{\text{orig}}} \|\boldsymbol{C} - \boldsymbol{C}^{\text{orig}}\|_2 \right) \tag{7}$$

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 τ times the nominal makespan value.

Two observations can be made with respect to Equation 7. (These observations are specific to this system only, and do not necessarily apply to the other example systems discussed later.) Let $\underline{\boldsymbol{C}^\star}$ be the value of $\boldsymbol{C}$ that minimizes $\|\boldsymbol{C} - \boldsymbol{C}^{\text{orig}}\|_2$.

(1) At the point $\boldsymbol{C}^\star$, the actual execution times for all applications are the same as the estimated times, except for the applications mapped on the machine that finishes last at $\boldsymbol{C}^\star$. This is because the finishing times are independent of each other, *and* the minimization is constrained only by the finishing time of one machine — the machine that finishes last.

(2) At the point $\boldsymbol{C}^\star$, the ETC errors for all applications mapped on the machine that finishes last are the same. This is because the weight of each such application towards determining the finishing time is the same (Equation 4).

Note that the calculation of the robustness metric above did not make any assumptions about the distribution of the estimation errors. In addition, note that $\rho_\mu(\Phi,\ \boldsymbol{C})$ has the units of $\boldsymbol{C}$, namely time.

4.2 The HiPer-D System

The second example derivation of the robustness metric is for a HiPer-D [10, 14] like system that maps 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 multi-tasking, executing the applications mapped to it in a round robin fashion. Similarly, a given network link is multi-tasked 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 modelled with a directed acyclic graph, shown in Figure 2. The

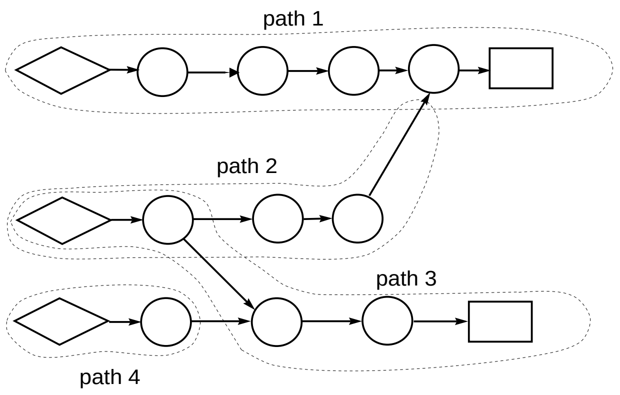


Figure 2: The DAG model for the applications (circles) and data transfers (arrows). The diamonds and rectangles denote sensors and actuators, respectively. The dashed lines enclose each path formed by the applications.

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 "trigger path") or at a multiple-input application (if it is an "update path"). Let $\underline{\mathcal{P}}$ be the set of all paths, and $\underline{\mathcal{P}_k}$ be the list of applications that comprise the k-th path. Note that an application may be present in multiple paths. As in Subsection 4.1, $\mathcal{A}$ 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 $\underline{T^{\mathrm{c}}_{ij}}$ be the computation time for application $\underline{a_i}$ mapped to machine $\underline{m_j}$. Also, let $\underline{T^{\mathrm{n}}_{ip}}$ be the time to send data from application a_i to application a_p. Because this analysis is being carried out for a specific mapping, the machine where a given application is mapped is known. It is assumed that a_i is mapped to m_j, and the machine subscript for T^{c}_{ij} 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, $\underline{L^{\max}_k}$. 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,

$$L_k = \sum_{\substack{i:\ a_i \in \mathcal{P}_k \\ p:\ (a_p \in \mathcal{P}_k) \wedge (a_p \in \mathcal{D}(a_i))}} \left[T^{\mathrm{c}}_i \ + \ T^{\mathrm{n}}_{ip} \right]. \tag{8}$$

It is desired that a given mapping μ 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 3, 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 Φ is given by

$$\Phi = \{T^{\mathrm{c}}_i \mid 1 \le i \le |\mathcal{A}|\} \bigcup \{T^{\mathrm{n}}_{ip} \mid (1 \le i \le |\mathcal{A}|) \wedge (\text{for } p \text{ where } a_p \in \mathcal{D}(a_i))\} \bigcup \{L_k \mid 1 \le k \le |\mathcal{P}|\} \tag{9}$$

This system is expected to operate under uncertain outputs from the sensors, requiring that the mapping μ be robust against unpredictable increases in the sensor outputs. Let $\underline{\lambda_z}$ 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, $\boldsymbol{\lambda}$, is the vector composed of the load values from all sensors. Let $\boldsymbol{\lambda}^{\text{orig}}$ be the initial value of $\boldsymbol{\lambda}$, and λ_i^{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 re-expressed as functions of $\boldsymbol{\lambda}$ as $T_i^{\text{c}}(\boldsymbol{\lambda})$ and $T_{ip}^{\text{n}}(\boldsymbol{\lambda})$, respectively. Then Equation 8 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 Equation 9 is given by

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

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

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

The robustness radius in Equation 10a 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. The robustness radii in Equations 10b and 10c are the similar values for the communications of application

a_x and the latency of path $\mathcal{P}_k$, respectively. The robustness metric, from Equation 2, is given by

$$\rho_\mu(\Phi,\ \boldsymbol{\lambda}) = \min_{\phi_i \in\ \Phi} \left(r_\mu(\phi_i,\ \boldsymbol{\lambda}) \right). \tag{11}$$

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 this section because the number of possible discrete values $\boldsymbol{\lambda}$ can take is infinite. However, because $\rho_\mu(\Phi,\ \boldsymbol{\lambda})$ should not have fractional values, one can take the floor of the right hand side in Equation 11. A different method for handling a discrete perturbation parameter is discussed in Subsection 4.3.

This research assumes that the optimization problems given in Equations 10a, 10b, and 10c can be solved to find the respective global minima. Note that an optimization problem of the form

$$\min_x f(x), \text{ subject to the constraint } g(x) = 0,$$

where $f(x)$ and $g(x)$ are both convex functions, can be easily solved to find the global minimum [5]. Because all norms are convex functions, the optimization problem posed in Equation 11 reduces to a convex optimization problem if $T^{\mathrm{c}}_x(\boldsymbol{\lambda})$ and $T^{\mathrm{n}}_{xy}(\boldsymbol{\lambda})$ are convex functions. Many commonly encountered complexity functions are convex. (A notable exception is $\log x$.) For example, all of the following functions are convex over the domain of positive real numbers ($x > 0$): e^{px} for $p \in \mathbf{R}$; x^p for $p \geq 1$; and $x \log x$. In addition, positive multiples of convex functions and sums of convex functions are also convex functions. Note that if the $T^{\mathrm{c}}_x(\boldsymbol{\lambda})$ and $T^{\mathrm{n}}_{xy}(\boldsymbol{\lambda})$ functions are not convex, then it is assumed that heuristic techniques can be used to find near-optimal solutions.

4.3 A System in Which Machine Failures Require Remapping

In many research efforts (e.g., [13, 18, 22]), flexibility of a mapping has been closely tied to its robustness, and is described as the quality of the mapping that can allow it to be changed easily into another mapping of comparable performance when system failures occur. This section briefly sketches the use of the four-step FePIA procedure to derive a robustness metric for systems where mapping changes become necessary due to dynamic events. In the example derivation analysis given below, it is assumed that re-mapping 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 re-assigned, $\underline{N^{\text{re-asgn}}}$, has to be less than $\underline{\tau_1}\%$ 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 mapping. Assume, without the loss of generality, that smaller values of J are more desirable. It is assumed that there is a specific re-mapping algorithm, which may not be the same as the original mapping algorithm. The re-mapping algorithm will re-assign the applications originally mapped to the failed machines to other machines, as well as re-assign some other applications if necessary. As in Subsection 4.1, $\mathcal{A}$ and $\mathcal{M}$ are the sets of applications and machines, respectively.

Following step 1 of the FePIA procedure,

$$\Phi = \{N^{\text{re-asgn}},\, J\}\,. \tag{12}$$

Step 2 requires that the perturbation parameter $\boldsymbol{\pi}_j$ be identified. Let that be $\underline{\boldsymbol{F}}$, a vector that indicates the identities of the machines that have failed. Specifically, $\boldsymbol{F} = [f_1\, f_2\, \cdots\, f_{|\mathcal{M}|}]^{\mathrm{T}}$ 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 re-mapping algorithm, as well as $\boldsymbol{F}$, and can be determined from the mapping produced by the re-mapping algorithm. Then $N^{\text{re-asgn}}$ and J can be re-expressed 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 re-assigned is less than the maximum tolerable number. Mathematically,

$$\left\{\boldsymbol{F} \middle| \ \left(N^{\text{re-asgn}}(\boldsymbol{F}) \leq \tau_1|\mathcal{A}|\right) \wedge \left(\exists \boldsymbol{F}^{(+1)} \quad N^{\text{re-asgn}}(\boldsymbol{F}^{(+1)}) > \tau_1|\mathcal{A}|\right)\right\}.$$

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

$$\left\{\boldsymbol{F} \middle| \ \left(J(\boldsymbol{F}) \leq \tau_2 J^{\text{orig}}\right) \wedge \left(\exists \boldsymbol{F}^{(+1)} \quad J(\boldsymbol{F}^{(+1)}) > \tau_2 J^{\text{orig}}\right)\right\}.$$

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

$$r_\mu(N^{\text{re-asgn}},\ \boldsymbol{F}) = \min_{\boldsymbol{F}:\ (N^{\text{re-asgn}}(\boldsymbol{F}) \leq \tau_1|\mathcal{A}|) \wedge (\exists \boldsymbol{F}^{(+1)}\ N^{\text{re-asgn}}(\boldsymbol{F}^{(+1)}) > \tau_1|\mathcal{A}|)} \|\boldsymbol{F} - \boldsymbol{F}^{\text{orig}}\|_2, \tag{13}$$

and

$$r_\mu(J,\ \boldsymbol{F}) = \min_{\boldsymbol{F}:\ (J(\boldsymbol{F}) \leq \tau_2 J^{\text{orig}}) \wedge (\exists \boldsymbol{F}^{(+1)}\ J(\boldsymbol{F}^{(+1)}) > \tau_2 J^{\text{orig}})} \|\boldsymbol{F} - \boldsymbol{F}^{\text{orig}}\|_2. \tag{14}$$

In Equations 13 and 14, the term $\|\boldsymbol{F} - \boldsymbol{F}^{\text{orig}}\|_2$ $(= \|\boldsymbol{F}\|_2)$ equals the square root of the number of the machines that fail. The square of the robustness radius in Equation

13 is the largest number of machines that can fail in any combination without causing the number of applications that have to be re-assigned to exceed $\tau_1|\mathcal{A}|$. Similarly, the square of the robustness radius in Equation 14 is the largest number of machines that can fail in any combination without causing the objective function in the re-mapped system to degrade beyond $\tau_2 J(\boldsymbol{F})$. The robustness metric, from Equation 2, is given by

$$\rho_\mu(\Phi,\ \boldsymbol{F}) = \min\big(r_\mu(N^{\text{re-asgn}},\ \boldsymbol{F}),\, r_\mu(J,\ \boldsymbol{F})\big). \tag{15}$$

As in Subsection 4.2, it is assumed here that the discrete optimization problems posed in Equations 13 and 14 can be solved for optimal or near-optimal solutions using combinatorial optimization techniques [21].

For this example, the $\underline{\ell_1\text{-norm}}$ (defined as $\sum_{r=1}^{n}|x_r|$ for a vector $\mathbf{x} = [x_1\, x_2\, \cdots\, x_n]^{\mathrm{T}}$) is more appropriate than the ℓ_2-norm for use in the robustness metric. In this case, it is natural to think of robustness in terms of the maximum allowable number of machines that can fail instead of the square root of the maximum allowable number of machines that can fail. As for the case with the ℓ_2-norm, it is assumed that the discrete optimization problems posed in Equations 13 and 14 can be solved for optimal or near-optimal solutions when the ℓ_1-norm is used.

5 Robustness Against Multiple Perturbation Parameters

Section 3 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 3. Specifically, this section develops an expression for the robustness radius for a single performance feature, ϕ_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 $\boldsymbol{\pi}_j$ have $\underline{n_{\boldsymbol{\pi}_j}}$ elements, and let $\star$ be the vector concatenation operator, so that

$$\boldsymbol{\pi}_1 \star \boldsymbol{\pi}_2 = [\ \pi_{11} \quad \pi_{12} \quad \cdots \quad \pi_{1n_{\pi_1}} \quad \pi_{21} \quad \pi_{22} \quad \cdots \quad \pi_{2n_{\pi_2}}\ ]^{\mathrm{T}}. \tag{16}$$

Let $\underline{\mathbf{P}}$ be a weighted concatenation of the vectors $\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \cdots, \boldsymbol{\pi}_{|\Pi|}$. That is, $\mathbf{P} = (\alpha_1 \times \boldsymbol{\pi}_1) \star (\alpha_2 \times \boldsymbol{\pi}_2) \star \cdots \star (\alpha_{|\Pi|} \times \boldsymbol{\pi}_{|\Pi|})$, where $\underline{\alpha_j}$ $(1 \le j \le |\Pi|)$ is a weighting constant that may be assigned by a system administrator or be based on the sensitivity of the system performance feature ϕ_i towards $\boldsymbol{\pi}_j$ (explained in detail later).

The vector $\mathbf{P}$ is analogous to the vector $\boldsymbol{\pi}_j$ discussed in Section 3. Parallel to the discussion in Section 3, one needs to identify the set of boundary values of $\mathbf{P}$. Let $\underline{f_i}$ be a function that maps $\mathbf{P}$ to ϕ_i. (Note that f_i could be independent of some $\boldsymbol{\pi}_j$.) For the single system feature ϕ_i being considered, such a set is given by

$$\left\{\mathbf{P} \,\middle|\, (f_i(\mathbf{P}) = \beta_i^{\min}) \bigvee (f_i(\mathbf{P}) = \beta_i^{\max})\right\}$$

Let $\underline{\mathbf{P}^{\mathrm{orig}}}$ be the assumed value of $\mathbf{P}$. In addition, let $\underline{\mathbf{P}^{\star}(\phi_i)}$ be, analogous to $\boldsymbol{\pi}_j^{\star}(\phi_i)$, the element in the set of boundary values such that the Euclidean distance from $\mathbf{P}^{\mathrm{orig}}$ to $\mathbf{P}^{\star}(\phi_i)$, $\|\mathbf{P}^{\star}(\phi_i) - \mathbf{P}^{\mathrm{orig}}\|_2$, is the smallest over all such distances from $\mathbf{P}^{\mathrm{orig}}$ to a point in the boundary set. Alternatively, the value $\|\mathbf{P}^{\star}(\phi_i) - \mathbf{P}^{\mathrm{orig}}\|_2$ gives the largest Euclidean distance that the variable $\mathbf{P}$ can move in *any* direction from an assumed value of $\mathbf{P}^{\mathrm{orig}}$ without exceeding the tolerable limits on ϕ_i. Parallel to the discussion in Section 3, let the distance $\|\mathbf{P}^{\star}(\phi_i) - \mathbf{P}^{\mathrm{orig}}\|_2$ be called the robustness radius, $r_\mu(\phi_i,\ \mathbf{P})$, of ϕ_i against $\mathbf{P}$. Mathematically,

$$r_\mu(\phi_i,\ \mathbf{P}) = \min_{\mathbf{P}:\,(f_i(\mathbf{P})=\beta_i^{\min}) \bigvee (f_i(\mathbf{P})=\beta_i^{\max})} \|\mathbf{P} - \mathbf{P}^{\mathrm{orig}}\|_2. \tag{17}$$

Extending for all $\phi_i \in \Phi$, the robustness of mapping μ with respect to the performance feature set Φ against the perturbation parameter set Π 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 α_j's is now discussed. Typically, $\boldsymbol{\pi}_1,\ \boldsymbol{\pi}_2,\ \cdots,\ \boldsymbol{\pi}_{|\Pi|}$ 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 ϕ_i, the magnitudes of α_j should indicate the relative sensitivities of ϕ_i to different $\boldsymbol{\pi}_j$'s. One way to accomplish the above goals is to set $\alpha_j = 1/r_\mu(\phi_i,\ \boldsymbol{\pi}_j)$. With this definition of α_j,

$$\mathbf{P} = \frac{\boldsymbol{\pi}_1}{r_\mu(\phi_i,\ \boldsymbol{\pi}_1)} \star \frac{\boldsymbol{\pi}_2}{r_\mu(\phi_i,\ \boldsymbol{\pi}_2)} \star \cdots \star \frac{\boldsymbol{\pi}_{|\Pi|}}{r_\mu(\phi_i,\ \boldsymbol{\pi}_{|\Pi|})}. \tag{18}$$

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

6 Experiments

The experiments in this section seek to establish the utility of the robustness metric in distinguishing between mappings 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 Subsection 4.1 and the HiPer-D system outlined in Subsection 4.2. Experiments were performed for a system with five machines and 20 applications. A total of 1000 mappings were generated by assigning a randomly chosen machine to each application, and then each mapping was evaluated with the robustness metric and the commonly used metric.

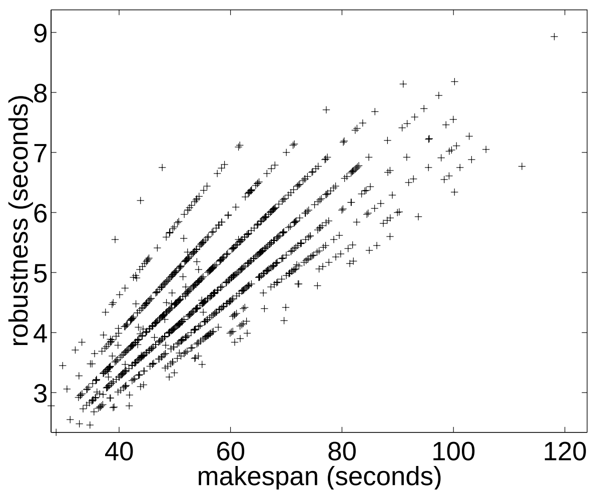
Independent Application Allocation System

For the system in Subsection 4.1, 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.

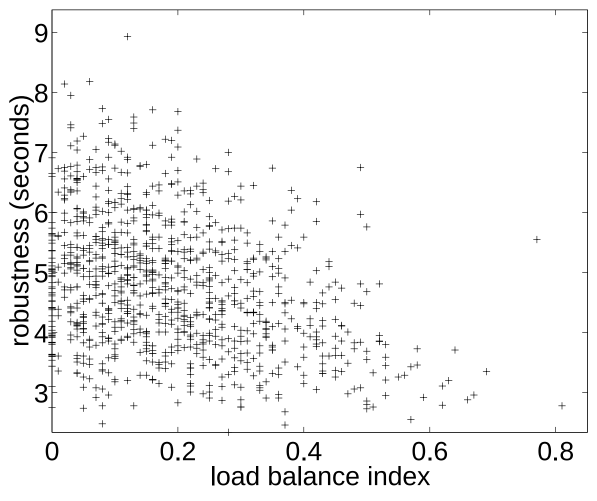
The mappings were evaluated for robustness, makespan, and <u>load balance index</u> (defined as the ratio of the finishing time of the machine that finishes first to the makespan). The larger the value of the load balance index, the more balanced the load (the largest value being 1). The tolerance, τ, was set to 20% (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 mapping means that the mapping 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.

Figure 3(a) shows the robustness of a mapping against its makespan. It can be seen that sharp differences exist in the robustness of some mappings that have very similar values of makespan. A similar conclusion can be drawn from the robustness against load balance index plot in Figure 3(b). Both of the above conclusions highlight the fact that the robustness metric can be used to differentiate between mappings that perform similarly with respect to two popular performance metrics.

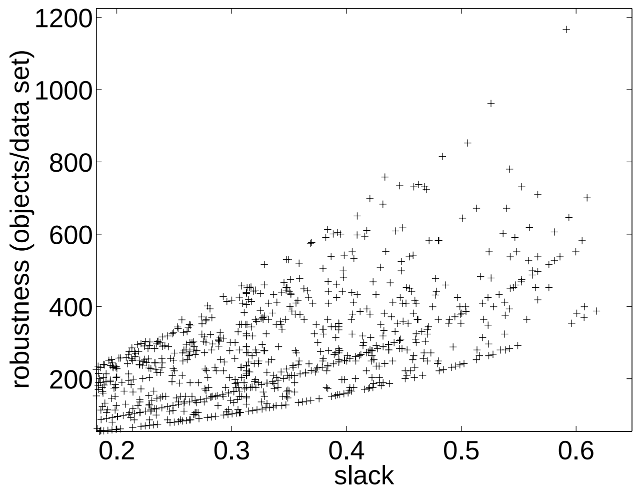
It can be seen in Figure 3(a) that some mappings are clustered into groups, such that for all mappings within a group, the robustness increases linearly with the makespan. This can be explained using Equation 6. Let $\underline{m(\boldsymbol{C})}$ be the machine that determines the makespan at $\boldsymbol{C}$. Let $n(m_j)$ be the number of applications mapped to machine m_j. If $m(\boldsymbol{C}^{\text{orig}})$ has the largest number of applications mapped to it, then it is also the machine that determines the robustness of the mapping (because it has the smallest robustness radius, Equation 6). Now consider the set $S_1(x)$ of mappings such that $x = n(m(\boldsymbol{C}^{\text{orig}})) = \max_{j:m_j \in \mathcal{M}} n(m_j)$ for each mapping in the set. For mappings in $S_1(x)$, the robustness is directly proportional to M^{orig} (Equation 6). Each distinct straight line in Figure 3(a) 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 mappings 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 mapping 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 $m(\boldsymbol{C}^{\text{orig}})$.



(a)



(b)



(c)

Figure 3: The plots of robustness against (a) makespan, (b) load balance index, and (c) slack for 1000 randomly generated mappings. The plots (a) and (b) correspond to the system in Subsection 4.1, and the plot (c) corresponds to the system in Subsection 4.2. For plot (a), generally, at any given value of makespan, there are a few mappings with significantly different values of robustness. The same is true for load balance index in plot (b) and slack in plot (c).

The HiPer-D System

For the model in Subsection 4.2, 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 4×10^{-5}, 3×10^{-5}, and 8×10^{-6}), and three actuators. The experiments made the following simplifying assumptions. The computation time function, $T_{ij}^{c}(\boldsymbol{\lambda})$, was assumed to be of the form $\sum_{1 \leq z \leq 3} \underline{b_{ijz}} \lambda_z$, 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 mappings were evaluated for robustness and "slack." In this context, a robustness value of x for a given mapping means that the mapping 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., [8, 18]) for mapping in parallel and distributed systems, where a mapping with a larger slack is considered better. In this study, slack is defined mathematically as follows. Let the <u>fractional</u> <u>value</u> of a given QoS attribute be the value of the attribute as a percentage of the maximum allowed value. Then the <u>percentage</u> <u>slack</u> for a given QoS attribute is the fractional value subtracted from 1. The <u>system-wide</u> <u>percentage</u> <u>slack</u> is the minimum value of percentage slack taken over all QoS constraints, and can be expressed mathematically as

$$\min\left(\min_{k:\,\mathcal{P}_k\in\mathcal{P}}\left(1-\frac{L_k(\boldsymbol{\lambda})}{L_k^{\max}}\right),\quad \min_{i:\,a_i\in\mathcal{A}}\left(1-\frac{\max\left(T_i^{\mathrm{c}}(\boldsymbol{\lambda}),\ \max\limits_{a_p\in\mathcal{D}(a_i)} T_{ip}^{\mathrm{n}}(\boldsymbol{\lambda})\right)}{1/R(a_i)}\right)\right).$$

Figure 3(c) shows the robustness of a mapping against its slack. It can be seen that while mappings with a larger slack are more robust in general, sharp differences exist in the robustness of some mappings that have very similar values of slack. In particular, examine the two mappings shown in Table 1. Although the slack values are approximately the same, the robustness of B is about 3.3 times that of A, and therefore B is a much better mapping considering its robustness against increases in sensor loads. That is, using slack as a measure of how much increase in sensor load a system can tolerate may cause system designers to grossly misjudge the systems capability. The results show that slack, as defined here, is not a good indicator of the robustness of the system as to how many objects per data set could be processed by the system before a QoS violation occurred. As can be seen in Figure 3(c), there is a set of mappings with slack values ranging from approximately 0.2 to approximately 0.5, but all these mappings have the same robustness value of approximately 250. These mappings are virtually indistinguishable from each other with respect to how many objects per data set could be processed by the system before a QoS violation occurred. The experiments in this section illustrate that, even for very simple computing environments, a commonly used measure of the availability of computing resources is not a reliable measure of the system's ability to maintain a desired QoS in the presence of uncertainty, and that an explicit measure of robustness should be used in the manner specified by the FePIA procedure presented here.

Table 1 also shows the actual sensor load values at which a throughput or latency constraint is reached for a given mapping. For reference, the computation time functions $T_{ij}^{\mathrm{c}}(\boldsymbol{\lambda})$ for this experiment are given in Table 1.

Table 1: For the system in Subsection 4.2, these two mappings perform similarly in terms of their slack values, but are significantly different in their robustness values. The initial sensor load values are $\lambda_1 = 962$, $\lambda_2 = 380$, and $\lambda_3 = 240$. The final sensor load values at which a throughput or latency constraint is reached for a given mapping are given in the table (λ_1^*, λ_2^*, and λ_3^*). The computation time function $T_{ij}^{c}(\boldsymbol{\lambda})$ for each application on the machine where it is mapped is also shown. For $T_{ij}^{c}(\boldsymbol{\lambda})$ functions, the number outside the parenthesis is the "multi-tasking factor," and equals $1.3n(m_j)$ where $n(m_j) \geq 2$ is the number of applications mapped to m_j. The function inside the parenthesis gives the complexity of the execution time as a function of $\boldsymbol{\lambda}$, and may be different for a given application mapped to different machines (e.g., as for a_2).

		mapping A	mapping B
robustness		353 objects/data set (based on Euclidean distance)	1166 objects/data set (based on Euclidean distance)
slack		0.5961	0.5914
$\lambda_1^*, \lambda_2^*, \lambda_3^*$		962, 380, 593	962, 1546, 240
application assignments	m_1	a_5, a_9, a_{12}, a_{17}, a_{20}	a_3, a_4, a_5, a_{17}, a_{18}, a_{20}
	m_2	a_6, a_{16}	a_2, a_{11}, a_{14}, a_{19}
	m_3	a_1, a_3, a_7	a_1, a_7, a_{13}
	m_4	a_2, a_4, a_{10}, a_{13}, a_{15}, a_{19}	a_9, a_{12}, a_{15}
	m_5	a_8, a_{11}, a_{14}, a_{18}	a_6, a_8, a_{10}, a_{16}
computation time functions	a_1	$3.90(4\lambda_3)$	$3.90(4\lambda_3)$
	a_2	$7.80(5\lambda_2)$	$5.20(2\lambda_2)$
	a_3	$3.90(6\lambda_1)$	$7.80(11\lambda_1)$
	a_4	$7.80(\lambda_1)$	$7.80(4\lambda_1 + 2\lambda_2)$
	a_5	$6.50(3\lambda_1 + \lambda_3)$	$7.80(3\lambda_1 + \lambda_3)$
	a_6	$2.60(\lambda_3)$	$5.20(\lambda_3)$
	a_7	$3.90(5\lambda_2)$	$3.90(5\lambda_2)$
	a_8	$5.20(6\lambda_2)$	$5.20(6\lambda_2)$
	a_9	$6.50(20\lambda_3)$	$3.90(3\lambda_3)$
	a_{10}	$7.80(5\lambda_2 + 7\lambda_3)$	$5.20(3\lambda_2 + 3\lambda_3)$
	a_{11}	$5.20(10\lambda_1 + 8\lambda_2 + 6\lambda_3)$	$5.20(10\lambda_1 + 4\lambda_2 + 8\lambda_3)$
	a_{12}	$6.50(26\lambda_1)$	$3.90(24\lambda_1)$
	a_{13}	$7.80(19\lambda_1 + 8\lambda_2)$	$3.90(23\lambda_1 + 6\lambda_2)$
	a_{14}	$5.20(11\lambda_1)$	$5.20(7\lambda_1)$
	a_{15}	$7.80(13\lambda_1 + 17\lambda_2 + 9\lambda_3)$	$3.90(13\lambda_1 + 17\lambda_2 + 9\lambda_3)$
	a_{16}	$2.60(2\lambda_2)$	$5.20(7\lambda_2)$
	a_{17}	$6.50(3\lambda_1 + 5\lambda_3)$	$7.80(3\lambda_1 + 5\lambda_3)$
	a_{18}	$5.20(3\lambda_1 + 19\lambda_2 + 11\lambda_3)$	$7.80(6\lambda_1 + 2\lambda_2 + 10\lambda_3)$
	a_{19}	$7.80(9\lambda_1 + 13\lambda_2)$	$5.20(4\lambda_1 + 8\lambda_2)$
	a_{20}	$6.50(3\lambda_1 + 14\lambda_2 + 18\lambda_3)$	$7.80(3\lambda_1 + 14\lambda_2 + 18\lambda_3)$

7 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 four-step procedure, the FePIA procedure, 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 mappings that perform similarly otherwise.

Acknowledgments: The authors thank Sahra Sedigh-Ali, Martin Nagel, and Safoora Sarvestani for their valuable comments. A preliminary version of the portions of this paper will be presented at the 17th Int'l Parallel and Distributed Processing Symposium, April 2003.

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," *The 2002 Int'l Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2002), Vol. II*, June 2002, pp. 519–530.

[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. of Science and Engineering*, Vol. 3, No. 3, Nov. 2000, pp. 195–207, invited.

[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. 26, 1993.

[4] L. Bölöni and D. C. Marinescu, "Robust scheduling of metaprograms," *J. of Scheduling*, Vol. 5, No. 5, Sep. 2002, pp. 395–412.

[5] S. Boyd and L. Vandenberghe, *Convex Optimization,* to appear in 2003, available at `http://www.stanford.edu/class/ee364/index.html#book`.

[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. of Parallel and Distributed Computing*, Vol. 61, No. 6, June 2001, pp. 810–837.

[7] J. M. Carlson and J. Doyle, "Complexity and robustness," *Proceedings of Nat'l Academy of Science (PNAS)*, Vol. 99, No. 1, Feb. 2002, pp. 2538–2545.

[8] A. Davenport, C. Gefflot, and J. Beck, "Slack-based techniques for robust schedules," *6th European Conf. on Planning (ECP-2001)*, Sep. 2001, pp. 7–18.

[9] J. DeVale, *High Performance Robust Computer Systems.* PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University, 2001.

[10] S. Gertphol, Y. Yu, S. B. Gundala, V. K. Prasanna, S. Ali, J.-K. Kim, A. A. Maciejewski, and H. J. Siegel, "A metric and mixed-integer-programming-based approach for resource allocation in dynamic real-time systems," *16th Int'l Parallel and Distributed Processing Symposium (IPDPS 2002)*, Apr. 2002.

[11] M. L. Ginsberg, A. J. Parkes, and A. Roy, "Supermodels and robustness," *15th Nat'l Conf. on Artificial Intelligence, American Association for Artificial Intelligence (AAAI)*, July 1998, pp. 334–339.

[12] S. D. Gribble, "Robustness in complex systems," *8th Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, May 2001, pp. 21–26.

[13] E. Hart, P. Ross, and J. Nelson, "Producing robust schedules via an artificial immune system," *The 1998 Int'l Conf. on Evolutionary Computing*, May 1998, pp. 464–469.

[14] 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.

[15] M. T. Jensen and T. K. Hansen, "Robust solutions to job shop problems," *The 1999 Congress on Evolutionary Computation*, July 1999, pp. 1138–1144.

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

[17] E. Jen, "Definitions," Santa Fe Institute Robustness Site, RS-2001-009, `http://discuss.santafe.edu/robustness`, posted 10-22-01.

[18] M. Jensen, "Improving robustness and flexibility of tardiness and total flowtime job shops using robustness measures," *J. of Applied Soft Computing*, Vol. 1, No. 1, June 2001, pp. 35–52.

[19] V. Leon, S. Wu, and R. Storer, "Robustness measures and robust scheduling for job shops," *IEE Transactions*, Vol. 26, No. 5, Sep. 1994, pp. 32–43.

[20] 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. of Parallel and Distributed Computing*, Vol. 59, No. 2, Nov. 1999, pp. 107–131.

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

[22] M. Sevaux and K. Sörensen, "Genetic algorithm for robust schedules," *8th Int'l Workshop on Project Management and Scheduling (PMS 2002)*, Apr. 2002, pp. 330–333.

[23] G. F. Simmons, *Calculus With Analytic Geometry, Second Edition*, McGraw-Hill, New York, NY, 1995.

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