Perspectives on Robust Resource Allocation for Heterogeneous Parallel and Distributed Systems

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

[†]University of Missouri-Rolla
Department of Electrical and Computer Engineering
Rolla, MO 65409-0040 USA
shoukat@umr.edu

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

Submitted to "Parallel Computing: Models, Algorithms, and Applications," CRC Press, the Computer and Information Science Series
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 some 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 chapter 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, (2) a procedure for deriving a robustness metric for an arbitrary system, (3) derivation of robustness metrics for three example distributed systems, and (4) design of static heuristics to determine a robust resource allocation for one of the example systems.

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, by the Colorado State University Center for Robustness in Computer Systems (funded by the Colorado Commission on Higher Education Technology Advancement Group through the Colorado Institute of Technology), and by the Colorado State University George T. Abell Endowment.

1 Introduction

This research focuses on the robustness of a resource allocation in parallel and distributed computing systems. What does robustness mean? Some dictionary definitions of robustness are: (a) strong and healthy, as in "a robust person" or "a robust mind," (b) sturdy or strongly formed, as in "a robust plastic," (c) suited to or requiring strength as in "a robust exercise" or "robust work," (d) firm in purpose or outlook as in "robust faith," (e) full-bodied as in "robust coffee," and (f) rough or rude as in "stories laden with robust humor." In the context of resource allocation in parallel and distributed computing systems, how is the concept of robustness defined?

The allocation of resources to computational applications in heterogeneous parallel and distributed computer systems should maximize some system performance measure. Allocation decisions and associated performance prediction are often based on estimated values of application parameters, whose actual values may differ; for example, the estimates may represent only average values, or the models used to generate the estimates may have limited accuracy. Furthermore, 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., [2, 3, 4, 9, 10, 26, 32, 31, 34, 38]). Thus, an important research problem is the development of resource management strategies that can guarantee a particular system performance given bounds on such uncertainties. A resource allocation is defined to be robust with respect to specified system performance features against perturbations (uncertainties) in specified system parameters if degradation in these features is constrained when limited perturbations occur. An important question then arises: given a resource allocation, what extent of departure from the assumed circumstances will cause a performance feature to be unacceptably degraded? That is, how robust is the system?

Any claim of robustness for a given system must answer these three questions: (a) what behavior of the system makes it robust? (b) what uncertainties is the system robust against? (c) quantitatively, exactly how robust is the system? To address these questions, we have designed a model for deriving the degree of robustness of a resource allocation, i.e., the maximum amount of collective uncertainty in system parameters within which a user-specified level of system performance can be guaranteed. The model will be presented and we will demonstrate its ability to select the most robust resource allocation from among those that otherwise perform similarly (based on the primary performance criterion). The model's use in static (off-line) resource allocation heuristics also will be demonstrated. In particular, we will describe a static heuristic designed to determine a robust resource allocation for one of the example distributed systems. In general, this work is applicable to different types of computing and communication environments, including parallel, distributed, cluster, grid, Internet, embedded, and wireless.

The rest of the chapter is organized as follows. Section 2 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. Three static heuristics to derive a robust resource allocation for an example distributed system are described in Section 7, and are evaluated through simulations in Section 8. A sampling of the related work is given in Section 9. Section 10 gives some possibilities of future work in this area, and Section 11 concludes the chapter. A glossary of the notation used in this chapter is given in Table 1. Note that, throughput this chapter, new symbols are underlined when they are introduced. Such underlining is not a part of the symbology.

Table 1: Glossary of notation.

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

2 Generalized Robustness Metric

This section proposes a general procedure, called $\underline{\text{FePIA}}$, for deriving a general robustness metric for any desired computing environment. The name for the above procedure stands for identifying (1) the performance features, (2) the perturbation parameters, (3) the impact of perturbation parameters on performance features, and (4) 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.

1. Describe quantitatively the requirement that makes the system robust. Based on this *robustness requirement*, determine the quality of service, 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 $\underline{\text{makespan}}$ (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 120% 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 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 $\langle \beta_i^{\min}, \beta_i^{\max} \rangle$ could be $\langle 0,\ 1.2 \times (\text{estimated makespan value}) \rangle$.

2. Identify all of the system and environment uncertainty 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 [10]), 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 estimated makespan) was based on the estimated application execution times. It is desired that the makespan be robust (stay within 120% of its estimated value) with respect to uncertainties in these estimated execution times.

   Mathematically, let Π be the set of perturbation 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. In general, representation of the perturbation parameters as separate elements of Π would be based on their nature or kind (e.g., message length variables in $\boldsymbol{\pi}_1$ and computation time variables in $\boldsymbol{\pi}_2$).

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 1(b) 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 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. The rest of this discussion will be developed assuming only one element in Π. The case where multiple perturbation parameters can affect a given ϕ_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 120% 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 Subsection 3.3 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 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 two-element perturbation vector $\boldsymbol{\pi}_j \in \mathbf{R}^2$. The curve shown in Figure 1 plots the set of boundary points $\{\boldsymbol{\pi}_j \vert\, f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}\}$ for a resource allocation $\underline{\mu}$. For this figure, the set of boundary points $\left\{\boldsymbol{\pi}_j \vert\, f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min}\right\}$ 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, 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 property 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 robustness 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 of resource allocation μ with respect to performance feature ϕ_i against the perturbation parameter $\boldsymbol{\pi}_j$.

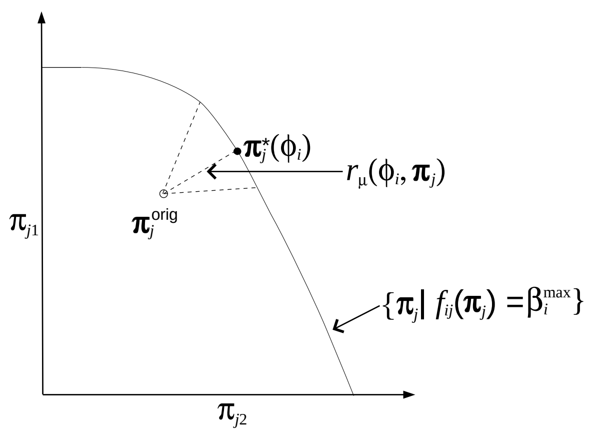
The robustness definition can be extended easily for all $\phi_i \in \Phi$ (see Figure 2). The robustness metric is simply the minimum of all robustness radii. Mathematically, let

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

Then, $\rho_\mu(\Phi,\ \boldsymbol{\pi}_j)$ is the *robustness metric of resource allocation μ with respect to the performance feature set Φ against the perturbation parameter $\boldsymbol{\pi}_j$.* Figure 2 shows a small system with a single two-element perturbation parameter, and only two performance features.

Even though the ℓ_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. Subsection 3.3 gives an example situation where the ℓ_1-norm is preferred over the ℓ_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.



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

Figure 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, $\left\{\boldsymbol{\pi}_j \,|\, f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min}\right\}$ is given by the points on the π_{j1}-axis and π_{j2}-axis.

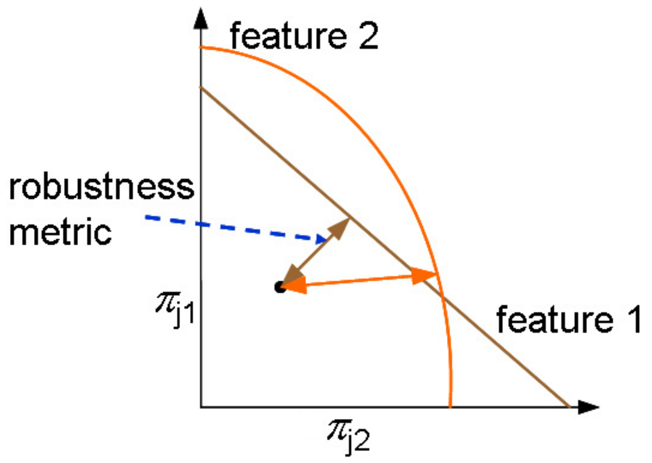


Figure 2: This figure shows a small system with a single perturbation parameter with elements π_{j1} and π_{j2}, and only two performance features. There is one robustness radius for each performance feature. The robustness metric is the smaller of the two robustness radii.

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 [12]. In this system, it is required that the makespan 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 (> 1) times the estimated makespan calculated using the assumed execution times. It is obvious that the larger the "factor," the larger the robustness. Assuming that ℓ_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. 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 resource allocation 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 allocated to it.

Assume that unknown inaccuracies in the ETC values are expected, requiring that the resource allocation μ 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 $\underline{\tau}$ (> 1) times its estimated value, $\underline{M^{\text{orig}}}$. The estimated value of the makespan is the value calculated assuming the ETC values are accurate (see Figure 3). 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, $\Phi = \{F_j | \; 1 \leq j \leq |\mathcal{M}|\}$.

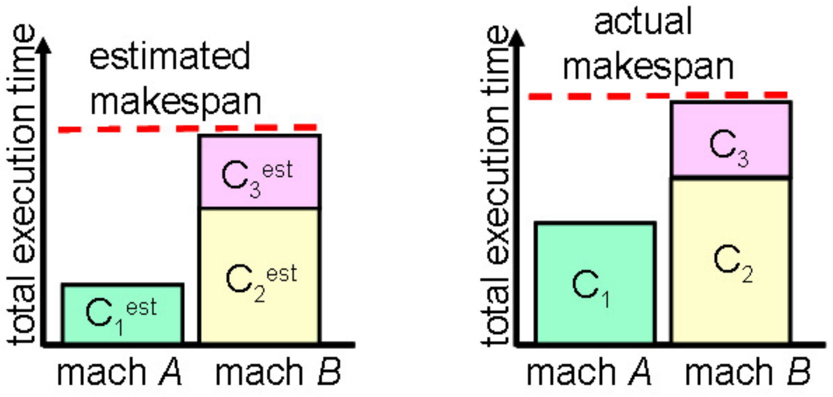


Figure 3: Three applications are executing in this 2-machine cluster. The estimated value of the makespan, calculated using the estimated computation times, is equal to the estimated finishing time of machine B in the graph on left. The actual makespan value, calculated using the actual computation times, is equal to the actual finishing time of machine B in the graph on right, and is more than the estimated value.

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 by resource allocation μ. 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,

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

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 $\left\{F_j(\boldsymbol{C}) = \tau M^{\text{orig}} \big| \ 1 \le j \le |\mathcal{M}|\right\}$.

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

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

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 τ times the estimated makespan value.

For example, assume only applications a_1 and a_2 have been assigned to machine j (depicted in Figure 4), and $\boldsymbol{C}$ has two components C_1 and C_2 that correspond to execution times of a_1 and a_2 on machine j, respectively. The term $F_j(\boldsymbol{C}^{\text{orig}})$ is the finishing time for machine j computed based on the ETC values of applications a_1 and a_2. Note that the right hand side in Equation 4 can be interpreted as the perpendicular distance from the point $\boldsymbol{C}^{\text{orig}}$ to the hyperplane described by the equation $\tau M^{\text{orig}} - F_j(\boldsymbol{C}) = 0$. Let $n(m_j)$ be the number of applications allocated to machine m_j. Using the point-to-plane distance formula [42], Equation 4 reduces to

$$r_\mu(F_j,\ \boldsymbol{C}) = \frac{\tau M^{\text{orig}} - F_j(\boldsymbol{C}^{\text{orig}})}{\sqrt{n(m_j)}} \tag{5}$$

The robustness metric, from Equation 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 τ times the estimated 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 [28] 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 [2], 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 allocated to it in a round robin fashion. Similarly, a given network link is multi-tasked among all

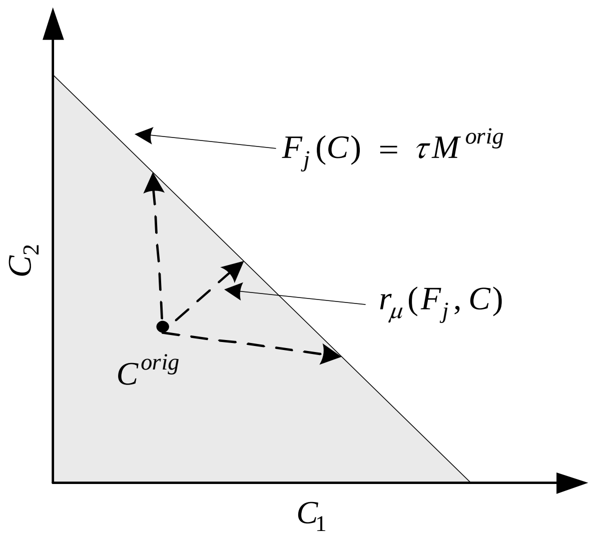


Figure 4: Some possible directions of increase of the perturbation parameter $\boldsymbol{C}$. The set of boundary points is given by $F_j(\boldsymbol{C}) = \tau M^{\text{orig}}$. The robustness radius $r_\mu(F_j, \boldsymbol{C})$ corresponds to the smallest increase that can reach the boundary. The shaded region represents the area of robust operation.

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 Figure 5. The

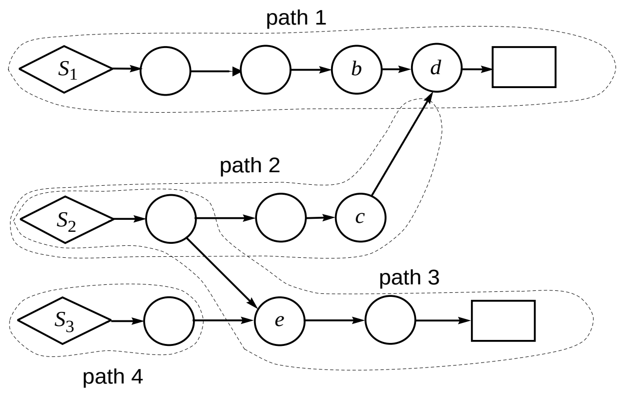


Figure 5: The DAG model for the applications (circles) and data transfers (arrows). The diamonds and rectangles denote sensors and actuators, respectively. The dashed lines enclose each path formed by the applications.

figure also shows a number of *paths* (enclosed by dashed lines) formed by the applications. A $\underline{\text{path}}$ is a chain of producer-consumer pairs that starts at a sensor (the $\underline{\text{driving sensor}}$) and ends at an actuator (if it is a $\underline{\text{trigger path}}$) or at a multiple-input application (if it is an $\underline{\text{update path}}$). In the context of Figure 5, 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. Example uses of this path model include defense, monitoring vital signs medical patients, recording scientific experiments, and surveillance for homeland security.

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 Subsection 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 <u>output data rate</u>. The <u>minimum throughput constraint</u> 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 $\mathcal{P}_k$ (see Figure 6). 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 $\underline{a_i}$ allocated 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 resource allocation, the machine where a given application is allocated is known. Let $\underline{\mathcal{D}(a_i)}$ be the set of successor applications of a_i. It is assumed that a_i is allocated 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. For the throughput constraint, $T^{\mathrm{c}}_{i} \leq 1/R(a_i)$ and $T^{\mathrm{n}}_{ip} \leq 1/R(a_i)$ for all $a_p \in \mathcal{D}(a_i)$.

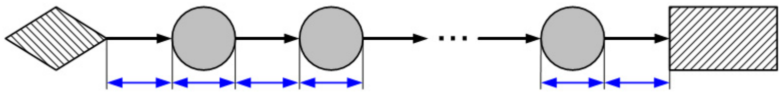


Figure 6: There is one throughput constraint for each application and one for each data transfer. It imposes a limit on each application's computation time and its data transfer time.

The <u>maximum end-to-end latency</u> 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, $\underline{L^{\max}_k}$ (see Figure 7). 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):

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

It is desired that a given resource allocation μ of the system be robust with respect to the satisfaction of two QoS attributes: the latency and throughput constraints. Following step 1

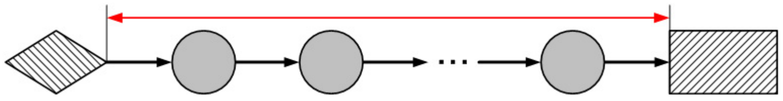


Figure 7: There is one latency constraint for each path. It imposes a limit on the time taken from the sensor output to the path "end point," which could be an actuator or a multiple-input application.

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 Φ is given by

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

This system is expected to operate under uncertain outputs from the sensors, requiring that the resource allocation 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 $\underline{\text{system}}$ $\underline{\text{workload}}$, $\underline{\boldsymbol{\lambda}}$, is the vector composed of the load values from all sensors. Let $\underline{\boldsymbol{\lambda}^{\mathrm{orig}}}$ be the initial value of $\boldsymbol{\lambda}$, and $\underline{\lambda_i^{\mathrm{orig}}}$ be the initial value of the i-th member of $\boldsymbol{\lambda}^{\mathrm{orig}}$. Following step 2 of FePIA, 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 $\underline{T_i^{\mathrm{c}}(\boldsymbol{\lambda})}$ and $\underline{T_{ip}^{\mathrm{n}}(\boldsymbol{\lambda})}$, respectively. Even though d is triggered only by b, its computation time depends on the outputs from both b and c. In general, $T_i^{\mathrm{c}}(\boldsymbol{\lambda})$ and $T_{ip}^{\mathrm{n}}(\boldsymbol{\lambda})$ 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 Figure 5 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 Equation 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 Equation 7 is given by

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

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

Specifically,

$$r_\mu(\phi_i,\ \boldsymbol{\lambda}) = \begin{cases} \min\limits_{\boldsymbol{\lambda}:\ T^{\text{c}}_x(\boldsymbol{\lambda})=1/R(a_x)} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{orig}}\|_2 & \text{if } \phi_i = T^{\text{c}}_x \quad (8\text{a}) \\ \min\limits_{\boldsymbol{\lambda}:\ T^{\text{n}}_{xy}(\boldsymbol{\lambda})=1/R(a_x)} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{orig}}\|_2 & \text{if } \phi_i = T^{\text{n}}_{xy} \quad (8\text{b}) \\ \min\limits_{\boldsymbol{\lambda}:\ L_k(\boldsymbol{\lambda})=L^{\max}_k} \|\boldsymbol{\lambda} - \boldsymbol{\lambda}^{\text{orig}}\|_2 & \text{if } \phi_i = L_k \quad (8\text{c}) \end{cases}$$

The robustness radius in Equation 8a 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 Equations 8b and 8c 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 $\rho_\mu(\Phi,\ \boldsymbol{\lambda}) = \min_{\phi_i \in\ \Phi} \left(r_\mu(\phi_i,\ \boldsymbol{\lambda})\right)$. 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 Equation 8 for the purpose of simplifying the illustration. A method for handling a discrete perturbation parameter is discussed in Subsection 3.3.

3.3 A System in Which Machine Failures Require Reallocation

In many research efforts (e.g., [27, 31, 38]), 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).

In our example, 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 $\underline{\tau_1}$% 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 (e.g., as done in [40]). As in Subsection 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 that be $\underline{\boldsymbol{F}}$, a vector that indicates the identities of the machines that have failed. Specifically, $\boldsymbol{F} = [f_1\, f_2\, \cdots\, f_{|\mathcal{M}|}]^{\text{T}}$ 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 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 reassigned is less than the maximum tolerable number. Mathematically,

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

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

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

Then, using Equation 1, one can find, 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 $\sum\limits_{r=1}^{n} |x_r|$ for a vector $\mathbf{x} = [x_1\, x_2\, \cdots\, x_n]^{\mathrm{T}}$) for use in the robustness metric. This is because, with the ℓ_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 ℓ_1-norm,

$$r_\mu(N^{\text{re-asgn}},\ \boldsymbol{F}) = \min_{\boldsymbol{F}:\ (N^{\text{re-asgn}}(\boldsymbol{F}) \le \tau_1|\mathcal{A}|) \wedge (\exists \boldsymbol{F}^{(+1)}\ N^{\text{re-asgn}}(\boldsymbol{F}^{(+1)}) > \tau_1|\mathcal{A}|)} \|\boldsymbol{F} - \boldsymbol{F}^{\text{orig}}\|_1, \tag{9}$$

and

$$r_\mu(J,\ \boldsymbol{F}) = \min_{\boldsymbol{F}:\ (J(\boldsymbol{F}) \le \tau_2 J^{\text{orig}}) \wedge (\exists \boldsymbol{F}^{(+1)}\ J(\boldsymbol{F}^{(+1)}) > \tau_2 J^{\text{orig}})} \|\boldsymbol{F} - \boldsymbol{F}^{\text{orig}}\|_1. \tag{10}$$

The robustness radius in Equation 9 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 Equation 10 is the largest number of machines that can fail in any combination without causing the objective function in the reallocated system to degrade beyond $\tau_2 J(\boldsymbol{F})$. The robustness metric, from Equation 2, is given by $\rho_\mu(\Phi,\ \boldsymbol{F}) = \min\left(r_\mu(N^{\text{re-asgn}},\ \boldsymbol{F}),\, r_\mu(J,\ \boldsymbol{F})\right)$. As in Subsection 3.2, it is assumed here that the discrete optimization problems posed in Equations 9 and 10 can be solved for optimal or near-optimal solutions using combinatorial optimization techniques [37].

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 [2]), 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 [2] and the references provided at the end of [2]), 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.

Our approach for handling multiple perturbation parameters is to concatenate 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, 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 $\diamondsuit$ be the vector concatenation operator, so that $\boldsymbol{\pi}_1 \diamondsuit \boldsymbol{\pi}_2 = [\ \pi_{11} \quad \pi_{12} \quad \cdots \quad \pi_{1n_{\pi_1}} \quad \pi_{21} \quad \pi_{22} \quad \cdots \quad \pi_{2n_{\pi_2}}\ ]^{\mathrm{T}}$. Let $\underline{\mathbf{P}}$ be a weighted concatenation of the vectors $\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \cdots, \boldsymbol{\pi}_{|\Pi|}$. That is, $\mathbf{P} = (\alpha_1 \times \boldsymbol{\pi}_1) \diamondsuit (\alpha_2 \times \boldsymbol{\pi}_2) \diamondsuit \cdots \diamondsuit (\alpha_{|\Pi|} \times \boldsymbol{\pi}_{|\Pi|})$, 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 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 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 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 2, 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,

$$r_\mu(\phi_i,\ \mathbf{P}) = \min_{\mathbf{P}:\, (f_i(\mathbf{P}) = \beta_i^{\min}) \bigvee (f_i(\mathbf{P}) = \beta_i^{\max})} \|\mathbf{P} - \mathbf{P}^{\mathrm{orig}}\|_2. \tag{11}$$

Extending for all $\phi_i \in \Phi$, the robustness of resource allocation 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. 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,

$$\mathbf{P} = \frac{\boldsymbol{\pi}_1}{r_\mu(\phi_i,\ \boldsymbol{\pi}_1)} \diamondsuit \frac{\boldsymbol{\pi}_2}{r_\mu(\phi_i,\ \boldsymbol{\pi}_2)} \diamondsuit \ \cdots \ \diamondsuit \frac{\boldsymbol{\pi}_{|\Pi|}}{r_\mu(\phi_i,\ \boldsymbol{\pi}_{|\Pi|})}. \tag{12}$$

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.

5 Computational Complexity

To calculate the robustness radius, one needs to solve the optimization problem posed in Equation 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 $\min_{x:\ l(x)=0} f(x)$ or $\min_{x:\ c(x)\geq 0} f(x)$ 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 [11].

Because all norms are convex functions [11], the optimization problem posed in Equation 1 reduces to a convex optimization problem if $f_{ij}(\boldsymbol{\pi}_j)$ is linear. One interesting problem with linear $f_{ij}(\boldsymbol{\pi}_j)$ is given in Subsection 3.1.

If $f_{ij}(\boldsymbol{\pi}_j)$ is concave, and the constraint "$f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\min}$" is irrelevant for some scenario (as it is for the system in Subsection 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, $\min\limits_{\boldsymbol{\pi}_j:\ (f_{ij}(\boldsymbol{\pi}_j)=\beta_i^{\max})} \|\boldsymbol{\pi}_j - \boldsymbol{\pi}_j^{\text{orig}}\|_2$ is equivalent to $\min\limits_{\boldsymbol{\pi}_j:\ (f_{ij}(\boldsymbol{\pi}_j)\geq\beta_i^{\max})} \|\boldsymbol{\pi}_j - \boldsymbol{\pi}_j^{\text{orig}}\|_2$. In such a case, the optimization problem would still be convex (and efficiently solvable) even if $f_{ij}(\boldsymbol{\pi}_j)$ were concave [11].

Similarly, if $f_{ij}(\boldsymbol{\pi}_j)$ is convex, and the constraint "$f_{ij}(\boldsymbol{\pi}_j) = \beta_i^{\max}$" is irrelevant for some scenario (e.g., for a 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 Equation 1 could still be solved for near-optimal solutions using heuristic approaches (some examples are given in [15]).

6 Demonstrating the Utility of the Proposed Robustness Metric

6.1 Overview

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 Subsection 3.1 and the HiPer-D system outlined in Subsection

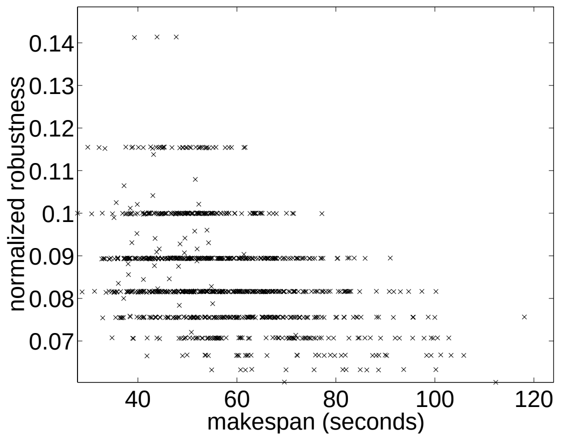
3.2. Experiments were performed for a system with five machines and 20 applications. A total of 1,000 resource allocations were generated by assigning a randomly chosen machine to each application, and then each resource allocation was evaluated with the robustness metric and the commonly used metric.

6.2 Independent Application Allocation System

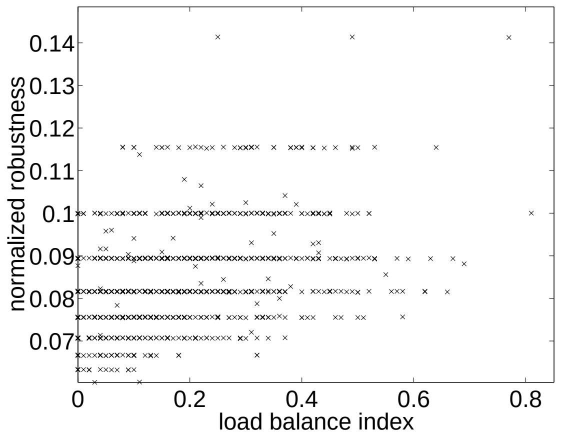
For the system in Subsection 3.1, the ETC values were generated by sampling a Gamma distribution. The mean was arbitrarily set to 10, the task heterogeneity was set to 0.7, and the machine heterogeneity was also set to 0.7 (the heterogeneity of a set of numbers is the standard deviation divided by the mean). See [5] 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, τ, was set to 120% (i.e., the actual makespan could be no more than 1.2 times the estimated 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 estimated value as long as the Euclidean norm of the errors is no larger than x seconds.

Figure 8(a) shows the "normalized robustness" of a resource allocation against its makespan. The normalized robustness equals the absolute robustness divided by the estimated makespan. A similar graph for the normalized robustness against the load balance index is shown in Figure 8(b). It can be seen in Figure 8 that some resource allocations are clustered into groups, such that for all resource allocations within a group, the normalized robustness remains constant as the estimated makespan (or load balance index) increases.



(a)



(b)

Figure 8: The plots of normalized robustness against (a) makespan, and (b) load balance index for 1,000 randomly generated resource allocations

The cluster of the resource allocations with the highest robustness has the feature that the machine with the largest finishing time has the smallest number of applications allocated to it (which is two for the experiments in Figure 8). The cluster with the smallest robustness has the largest number, 11, of applications allocated to the machine with the largest finishing time. The intuitive explanation for this behavior is that the larger the number of applications allocated to a machine, the more the 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 ℓ_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 (Figure 8(b)). In fact, it is possible to find a set of resource allocations that have very similar values of the makespan, and very similar values of the load balance index, but with very different values of the robustness. These observations highlight the fact that the information given by the robustness metric could not be obtained from two popular performance metrics. A typical way of using this robustness measure in a resource allocation algorithm would be to have a bi-objective optimization criterion where one would attempt to optimize makespan while trying to maximize robustness.

The clustering seen in Figure 8 can be explained using Equation 5. Consider a machine m_g that finishes last for a given resource allocation. Then the finishing time for m_g is equal to the makespan, M^{orig}, of that resource allocation. Call m_g the $\underline{\text{makespan}}$ $\underline{\text{machine}}$. Now $F_g(\boldsymbol{C}^{\text{orig}}) = M^{\text{orig}}$. From Equation 5 for m_g, we have

$$\begin{aligned} r_\mu(F_g,\ \boldsymbol{C}) &= \frac{\tau M^{\text{orig}} - M^{\text{orig}}}{\sqrt{n(m_g)}} \\ \frac{r_\mu(F_g,\ \boldsymbol{C})}{M^{\text{orig}}} &= \frac{\tau - 1}{\sqrt{n(m_g)}} \end{aligned} \tag{13}$$

The LHS in Equation 13 is the normalized robustness radius for m_g. If the makespan machine is also equal to the $\underline{\text{robustness}}$ $\underline{\text{machine}}$, i.e, m_g is such that $g = \text{argmin}_j(r_\mu(F_j,\ \boldsymbol{C}))$, then the LHS equals to the overall normalized robustness plotted in Figure 8. Now consider a set of resource allocations that have different makespans but all share one common feature: for all, the makespan machine is the same as the robustness machine and the number of applications on the makespan machine is 2. These resource allocations will give rise to the top cluster in Figure 8. The lower clusters are for higher values of $n(m_g)$. Note that "inter-cluster" distance decreases as $n(m_g)$ increases, as indicated by Equation 13. The outlying points belong to the resource allocations for which the makespan and robustness machines are different.

6.3 The HiPer-D System

For the model in Subsection 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, 1,250]. 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, $T_i^{\text{c}}(\boldsymbol{\lambda})$, was assumed to be linear in $\boldsymbol{\lambda}$. Specifically, for real number $\underline{b_{iz}}$, $T_i^{\mathrm{c}}(\boldsymbol{\lambda})$ was assumed to be of the form $\sum_{1 \le z \le 3} \underline{b_{iz}} \lambda_z$. In case, there was no route from the z-th sensor to application a_i, we set b_{iz} to 0 to ensure $T_i^{\mathrm{c}}(\boldsymbol{\lambda})$ would not depend on the load from the z-th sensor. If there was a route from the z-th sensor to application a_i, b_{iz} was sampled from a Gamma distribution with a mean of 10 and a task heterogeneity of 0.7. 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., [18, 31]) for resource allocation in parallel and distributed systems, where a resource allocation with a larger slack is considered to be more "robust" in a sense that it can better tolerate additional load. In this study, slack is defined mathematically as follows. Let the $\overline{\text{fractional}}$ $\overline{\text{value}}$ of a given performance feature be the value of the feature as a percentage of the $\overline{\text{maximum}}$ allowed value. Then the $\underline{\text{percentage}}$ $\underline{\text{slack}}$ for a given feature is the fractional value subtracted from 1. Intuitively, this is the percentage increase in this feature possible before reaching the maximum allowed value (see Figure 9). The $\underline{\text{system-wide}}$ $\underline{\text{percentage}}$ $\underline{\text{slack}}$ is the minimum value of percentage slack taken over all performance features, and can be expressed mathematically as

$$\min\left(\min_{k:\,\mathcal{P}_k \in \mathcal{P}}\left(1 - \frac{L_k(\boldsymbol{\lambda})}{L_k^{\max}}\right),\quad \min_{i:\,a_i \in \mathcal{A}}\left(1 - \frac{\max\left(T_i^{\mathrm{c}}(\boldsymbol{\lambda}),\ \max_{a_p \in \mathcal{D}(a_i)} T_{ip}^{\mathrm{n}}(\boldsymbol{\lambda})\right)}{1/R(a_i)}\right)\right). \tag{14}$$

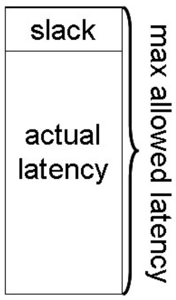


Figure 9: Slack in latency is the percentage increase in the actual latency possible before reaching the maximum allowed latency value.

Recall that all resource allocations meet the primary performance criteria of obeying all of the throughput and latency constraints. Thus we can select a resource allocation that, in addition to meeting primary criteria, is most robust.

Figure 10 shows the normalized robustness of a resource allocation against its slack. For this system, the normalized robustness equals the absolute robustness divided by $\|\boldsymbol{\lambda}^{\mathrm{orig}}\|_2$. 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, Equation 14, 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 resource allocations 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.

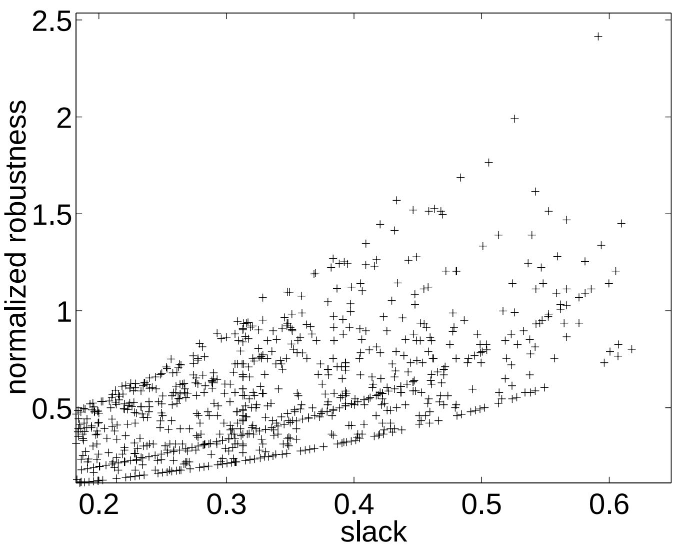


Figure 10: The plot of normalized robustness against slack for 1,000 randomly generated resource allocations

7 Designing Static Heuristics to Optimize Robustness for the HiPer-D System

7.1 Overview

The rest of this chapter describes the design and development of several static resource allocations heuristics for the HiPer-D system described in Subsection 3.2. The resource allocation problem has been shown, in general, to be NP-complete [16, 21, 29]. Thus, the development of heuristic techniques to find near-optimal resource allocation is an active area of research, e.g., [1, 7, 8, 12, 13, 20, 22, 24, 33, 35, 36, 39, 41, 44, 45, 46].

Static resource allocation is performed when the applications are mapped in an off-line planning phase such as in a production environment. Static resource allocation techniques take a set of applications, a set of machines, and generate a resource allocation. These heuristics determine a resource allocation off-line, and must use estimated values of application computation times and inter-application communication times.

Recall that for a HiPer-D system there are usually a number of QoS constraints that must be satisfied. A heuristic failure occurs if the heuristic cannot find a resource allocation that allows the system to meet its throughput and latency constraints. The system is expected to operate in an uncertain environment where the workload, i.e., the load presented by the set of sensors, is likely to change unpredictably, possibly invalidating a resource allocation that was based on the initial workload estimate. The focus here is on designing a static heuristic that: (a) determines an optimally *robust* resource allocation, i.e., a resource allocation that maximizes the allowable increase in workload until a run-time reallocation of resources is required to avoid a QoS violation, and (b) has a very low *failure rate*.

We propose a heuristic that performs well with respect to the failure rates and robustness to unpredictable workload increases. This heuristic is, therefore, very desirable for systems where low failure rates can be a critical requirement and where unpredictable circumstances can lead to unknown increases in the system workload.

7.2 Computation and Communication Models

7.2.1 Computation Model

This subsection summarizes the computation model for this system as originally described in [2]. For an application, the estimated time to compute a given data set depends on the load presented by the data set, and the machine executing the application. Let $\underline{C_{ij}(\boldsymbol{\lambda})}$ be the estimated time to compute for application a_i on m_j for a given workload (generated by $\boldsymbol{\lambda}$) when a_i is the only application executing on m_j. This research assumes that $C_{ij}(\boldsymbol{\lambda})$ is a function known for all i, j, and $\boldsymbol{\lambda}$.

The effect of multitasking on the computation time of an application is accounted for by assuming that all applications mapped to a machine are processing data continuously. Let $\underline{N_j}$ be the number of applications executing on machine m_j. Let $\underline{T^{\mathrm{c}}_{ij}(\boldsymbol{\lambda})}$ be the computation time for a_i on machine m_j when a_i shares this machine with other applications[1]. If the overhead due to context switching is ignored, $T^{\mathrm{c}}_{ij}(\boldsymbol{\lambda})$ will be $N_j \times C_{ij}(\boldsymbol{\lambda})$.

However, the overhead in computation time introduced by context switching may not be trivial in a round-robin scheduler. Such an overhead depends on the estimated-time-to-compute value of the application on the machine, and the number of applications executing on the machine. Let $\underline{O^{\mathrm{cs}}_{ij}(\boldsymbol{\lambda})}$ be the context switching overhead incurred when application a_i executes a given workload on machine m_j. Let $\underline{T^{\mathrm{cs}}_j}$ be the time m_j needs in switching the execution from one application to another. Let $\underline{T^{q}_j}$ be the size (quantum) of the time slice

[1]The function $T^{\mathrm{c}}_i(\boldsymbol{\lambda})$ used in Subsection 3.2 is essentially the same as the function $T^{\mathrm{c}}_{ij}(\boldsymbol{\lambda})$. The subscript j was absent in the former because the identity of the machine where application a_i was mapped was not relevant to the discussion in Subsection 3.2.

given by m_j to each application in the round-robin scheduling policy used on m_j. Then,

$$O^{\text{cs}}_{ij}(\boldsymbol{\lambda}) = \begin{cases} 0 & \text{if } N_j = 1 \\ \dfrac{C_{ij}(\boldsymbol{\lambda}) \times N_j}{T^q_j} \times T^{\text{cs}}_j & \text{if } N_j > 1 \end{cases}$$

$T^{\text{c}}_{ij}(\boldsymbol{\lambda})$ can now be stated as,

$$T^{\text{c}}_{ij}(\boldsymbol{\lambda}) = \begin{cases} C_{ij}(\boldsymbol{\lambda}) & \text{if } N_j = 1 \\ C_{ij}(\boldsymbol{\lambda}) \times N_j \left(1 + \dfrac{T^{\text{cs}}_j}{T^q_j}\right) & \text{if } N_j > 1 \end{cases} \tag{15}$$

7.2.2 Communication Model

This section develops an expression for the time needed to transfer the output data from a source application to a destination application at a given load. This formulation was originally given in [2]. Let $M_{ip}(\boldsymbol{\lambda})$ be the size of the message data sent from application a_i to a destination application a_p at the given load. Let $m(a_i)$ be the machine on which a_i is mapped. Let $T^{\text{n}}_{ip}(\boldsymbol{\lambda})$ be the transfer time, i.e., the time to send data from application a_i to application a_p at the given load.

A model for calculating the communication times should identify the various steps involved in effecting a data transfer. The two steps identified in [23] for communication in a similar domain are the communication setup time and the queuing delay. These steps contribute to the overall communication time to different extents depending on the intended communications environment.

Communication Setup Time. Before data can be transferred from one machine to another, a communication setup time is required for setting up a *logical* communication channel between the sender application and the destination application. Once established, a logical communication channel is torn down only when the sending or receiving application is finished executing. Because this study considers continuously executing applications (because the sensors continually produce data), the setup time will be incurred only once, and will, therefore, be amortized over the course of the application execution. Hence, the communication setup time is ignored in this study.

Queuing Delay. A data packet is queued twice enroute from the source machine to the destination machine. First, the data packet is queued in the output buffer of the source machine, where it waits to use the communication link from the source machine to the switch. The second time, the data packet is queued in the output port of the switch, where it waits to use the communication link from the switch to the destination machine. The switch has a separate output port for each machine in the system.

The queuing delay at the sending machine is modeled by assuming that the bandwidth of the link from the sending machine to the switch is shared equally among all data transfers originating at the sending machine. This will underestimate the bandwidth available for each transfer because it assumes that all of the other transfers are always being performed. Similarly, the queuing delay at the switch is modeled by assuming that the bandwidth of the link from the switch to the destination machine is equally divided among all data transfers originating at the switch and destined for the destination machine. Let $B(m(a_i), \text{swt})$ be the bandwidth (in bytes per unit time) of the communication link between $m(a_i)$ and the switch, and $\underline{B(\text{swt}, m(a_p))}$ be the bandwidth (in bytes per unit time) of the communication link between the switch and $m(a_p)$. The abbreviation "swt" stands for "switch." Let $\underline{N^{\text{ct}}(m(a_i), \text{swt})}$ be the number of data transfers using the communication link from $m(a_i)$ to the switch. The superscript "ct" stands for "contention." Let $\underline{N^{\text{ct}}(\text{swt}, m(a_p))}$ be the number of data transfers using the communication link from the switch to $m(a_p)$. Then, $T^{\text{n}}_{ip}(\boldsymbol{\lambda})$, the time to transfer the output data from application a_i to a_p, is given by:

$$T^{\text{n}}_{ip}(\boldsymbol{\lambda}) = M_{ip}(\boldsymbol{\lambda}) \times \left(\frac{N^{\text{ct}}(m(a_i), \text{swt})}{B(m(a_i), \text{swt})} + \frac{N^{\text{ct}}(\text{swt}, m(a_p))}{B(\text{swt}, m(a_p))} \right) \tag{16}$$

The above expression can also accommodate the situations when a sensor communicates with the first application in a path, or when the last application in a path communicates with an actuator. The driving sensor for $\mathcal{P}_k$ can be treated as a "pseudo-application" that has a zero computation time and is already mapped to an imaginary machine, and, as such, can be denoted by $\underline{\alpha_{k,0}}$. Similarly, the actuator receiving data from $\mathcal{P}_k$ can also be treated as a pseudo-application with a zero computation time, and will be denoted by $\underline{\alpha_{k,|\mathcal{P}_k|+1}}$. Accordingly, for these cases: $M_{ip}(\boldsymbol{\lambda})$ corresponds to the size of the data set being sent from the sensor; $B(m(\alpha_{k,0}), \text{swt})$ is the bandwidth of the link between the sensor and the switch; and $B(\text{swt}, m(\alpha_{k,|\mathcal{P}_k|+1}))$ corresponds to the bandwidth of the link between the switch and the actuator. Similarly, $N^{\text{ct}}(m(\alpha_{k,0}), \text{swt})$ and $N^{\text{ct}}(\text{swt}, \alpha_{k,|\mathcal{P}_k|+1}))$ both are 1. In the situation where $m(a_i) = m(a_p)$, one can interpret $N^{\text{ct}}(m(a_i), \text{swt})$ and $N^{\text{ct}}(\text{swt}, m(a_p))$ both as 0; $T^{\text{n}}_{ip}(\boldsymbol{\lambda}) = 0$ in that case.

7.3 Heuristic Descriptions

7.3.1 Overview

This section develops three greedy heuristics for the problem of finding an initial static allocation of applications onto machines to maximize $\rho_\mu(\Phi,\ \boldsymbol{\lambda})$, where Φ is as defined in Equation 7. From this point on, for the sake of simplicity we will denote $\rho_\mu(\Phi,\ \boldsymbol{\lambda})$ by $\underline{\Delta\boldsymbol{\Lambda}}$. Greedy techniques perform well in many situations, and have been well-studied (e.g., [29]). One of the heuristics, Most Critical Task First (MCTF), is designed to work well in heterogeneous systems where the throughput constraints are more stringent than the latency constraints. Another heuristic, the Most Critical Path First (MCPF) heuristic, is designed to work well in heterogeneous systems where the latency constraints are more stringent than the throughput constraints.

It is important to note that these heuristics use the $\Delta\boldsymbol{\Lambda}$ value to guide the heuristic search; however, the procedure given in Section 3.2 (Equation 8) for calculating $\Delta\boldsymbol{\Lambda}$ assumes that a complete resource allocation of all applications is known. During the course of the execution of the heuristics, not all applications are mapped. In these cases, for calculating $\Delta\boldsymbol{\Lambda}$, the heuristics assume that each such application a_i is mapped to the machine where its computation time is smallest over all machines, and that a_i is using 100% of that machine. Similarly for communications where either the source or the destination application (or both) are unmapped, it is assumed that the data transfer between the source and destination occurs over the highest speed communication link available in the network, and that the link is 100% utilized by the data transfer. With these assumptions, $\Delta\boldsymbol{\Lambda}$ is calculated and used in any step of a given heuristic.

Before discussing the heuristics, some additional terms are now defined. Let $\underline{\Delta\mathbf{\Lambda}^{\mathrm{T}}}$ be the robustness of the resource allocation when only throughput constraints are considered, i.e, all latency constraints are ignored. Similarly, let $\underline{\Delta\mathbf{\Lambda}^{\mathrm{L}}}$ be the robustness of the resource allocation when only latency constraints are considered. In addition, let $\underline{\Delta\mathbf{\Lambda}_{ij}^{\mathrm{T}}}$ be the robustness of the assignment of a_i to machine m_j with respect to the throughput constraint, i.e., it is the largest increase in load in any direction from the initial value that does not cause a throughput violation for application a_i, either for the computation of a_i on machine m_j or for the communications from a_i to any of its successor applications. Similarly, let $\underline{\Delta\mathbf{\Lambda}_{k}^{\mathrm{L}}}$ be the robustness of the assignment of applications in $\mathcal{P}_k$ with respect to the latency constraint, i.e., it is the largest increase in load in any direction from the initial value that does not cause a latency violation for the path $\mathcal{P}_k$.

7.3.2 Most Critical Task First Heuristic

The Most Critical Task First Heuristic (MCTF) heuristic makes one application to machine assignment in each iteration. Each iteration can be split into two phases. Let $\underline{\mathcal{M}}$ be the set of machines in the system. Let $\underline{\Delta\mathbf{\Lambda}^{*}(a_i, m_j)}$ be the value of $\Delta\mathbf{\Lambda}$ *if* application a_i is mapped on m_j. Similarly, let $\underline{\Delta\mathbf{\Lambda}^{\mathrm{T}*}(a_i, m_j)}$ be the value of $\Delta\mathbf{\Lambda}_{ij}^{\mathrm{T}}$ *if* application a_i is mapped on m_j. In the first phase, each unmapped application a_i is paired with its "best" machine m_j such that

$$m_j = \underset{m_k \in \mathcal{M}}{\operatorname{argmax}} (\Delta\mathbf{\Lambda}^{*}(a_i, m_k)). \tag{17}$$

(Note that $\operatorname{argmax}_x f(x)$ returns the value of x that maximizes the function $f(x)$. If there are multiple values of x that maximize $f(x)$, then $\operatorname{argmax}_x f(x)$ returns the set of all those values.) If the RHS in Equation 17 returns a set of machines, $\underline{G(a_i)}$, instead of a unique machine, then $m_j = \operatorname{argmax}_{m_k \in G(a_i)}(\Delta\mathbf{\Lambda}^{\mathrm{T}*}(a_i, m_k))$, i.e., the individual throughput constraints are used to break ties in the overall system-wide measure. If $\Delta\mathbf{\Lambda}^{*}(a_i, m_j) < 0$, this heuristic cannot find a resource allocation. The first phase does not make an application to machine assignment; it only establishes application-machine pairs (a_i, m_j) for all unmapped applications a_i.

The second phase makes an application to machine assignment by selecting one of the (a_i, m_j) pairs produced by the first phase. This selection is made by determining the most "critical" application (the criterion for this is explained later). The method used to determine this assignment in the first iteration is totally different from that used in the subsequent iterations.

Consider the motivation of the heuristic for the special first iteration. Before the first iteration of the heuristic, all applications are unmapped, and the system resources are entirely unused. With the system in this state, the heuristic selects the pair (a_x, m_y) such that

$$(a_x, m_y) = \underset{\substack{(a_i, m_j) \text{ pairs from} \\ \text{the first phase}}}{\operatorname{argmin}} (\Delta\mathbf{\Lambda}^{*}(a_i, m_j)).$$

That is, from all of the (application, machine) pairs chosen in the first step, the heuristic chooses the pair (a_x, m_y) that leads to the smallest robustness. The application a_x is then assigned to the machine m_y. It is likely that if the assignment of this application is postponed, it might have to be assigned to a machine where its maximum allowable increase in the system load is even smaller. (The discussion above does not imply that an optimal resource allocation must contain the assignment of a_x on m_y.) Experiments conducted in this study have shown that the special first iteration significantly improves the performance.

The criterion used to make the second phase application to machine assignment for iterations 2 to $|\mathcal{A}|$ is different from that used in iteration 1, and is now explained. The intuitive goal is to determine the (a_i, m_j) pair, which if not selected, may cause the most future "damage," i.e., decrease in $\Delta\mathbf{\Lambda}$. Let $\underline{\mathcal{M}^{a_i}}$ be the ordered list, $\langle m_1^{a_i},\ m_2^{a_i},\ \cdots,\ m_{|\mathcal{M}|}^{a_i}\rangle$, of machines such that $\Delta\mathbf{\Lambda}^*(a_i,\ m_x^{a_i}) \geq \Delta\mathbf{\Lambda}^*(a_i,\ m_y^{a_i})$ if $x < y$. Note that $m_1^{a_i}$ is the same as a_i's "best" machine. Let $\underline{v}$ be an integer such that $2 \leq v \leq |\mathcal{M}|$, and let $\underline{r(a_i, v)}$ be the percentage decrease in $\Delta\mathbf{\Lambda}^*(a_i, m_j)$ if a_i is mapped on $m_v^{a_i}$ (its v-th best machine) instead of $m_1^{a_i}$, i.e.,

$$r(a_i, v) = \frac{\Delta\mathbf{\Lambda}^*(a_i,\ m_1^{a_i}) - \Delta\mathbf{\Lambda}^*(a_i,\ m_v^{a_i})}{\Delta\mathbf{\Lambda}^*(a_i,\ m_1^{a_i})}.$$

Additionally, let $\underline{T(a_i, 2)}$ be defined such that,

$$T(a_i, 2) = \frac{\Delta\mathbf{\Lambda}^{\mathrm{T}*}(a_i,\ m_1^{a_i}) - \Delta\mathbf{\Lambda}^{\mathrm{T}*}(a_i,\ m_2^{a_i})}{\Delta\mathbf{\Lambda}^{\mathrm{T}*}(a_i,\ m_1^{a_i})}.$$

Then, in all iterations other than the first iteration, MCTF maps the $\underline{\text{most}}$ $\underline{\text{critical}}$ application, where the most critical application is found using the pseudo-code in Figure 11. The program in Figure 11 takes the set of (a_i, m_j) pairs from the first phase of MCTF as its input. Then for each pair (a_i, m_j) it determines the percentage decrease in $\Delta\mathbf{\Lambda}^*(a_i, m_j)$ if a_i is mapped on its second best machine instead of its best machine. Then, once this information is calculated for all of the pairs obtained from the first phase, the program in Figure 11 chooses the pair for which the percentage decrease in $\Delta\mathbf{\Lambda}^*(a_i, m_j)$ is the largest. It may very well be the case that all pairs are alike, i.e., all have the same percentage decrease in $\Delta\mathbf{\Lambda}^*(a_i, m_j)$ when the program considers their second best and the best machines. In such a case, the program makes the comparisons again, however using the third best and the best machines this time. This continues until a pair is found that has the largest percentage decrease in $\Delta\mathbf{\Lambda}^*(a_i, m_j)$ or until all possible comparisons have been made and still there is no unique "winner." If we do not have a unique winner even after having compared the $|\mathcal{M}|$-th best machine with the best machine, we use $T(a_i, 2)$ to break the ties, i.e., we choose the pair that maximizes $T(a_i, 2)$. If, even that is not unique, we arbitrarily select one pair. This method is explained in detail in Figure 11. The technique shown in Figure 11 builds on the idea of the Sufferage heuristic given in [35].

7.3.3 Two-Phase Greedy Heuristic

This research also proposes a modified version of the Min-min heuristic. Variants of the Min-min heuristic (first presented in [29]) have been studied, e.g., [6, 12, 35, 46], and have been seen to perform well in the environments for which they were proposed. $\underline{\text{Two-Phase Greedy}}$ ($\underline{\text{TPG}}$), a Min-min style heuristic for the environment discussed in this research, is shown in Figure 12.

Like MCTF, the TPG heuristic makes one application to machine assignment in each iteration. Each iteration can be split into two phases. In the first phase, each unmapped application a_i is paired with its "best" machine m_j such that $m_j = \text{argmax}_{m_k \in \mathcal{M}}(\Delta\mathbf{\Lambda}^*(a_i, m_k))$.

```
1: initialize: $v = 2$; $\mathcal{F}$ = the set of $(a_i, m_j)$ pairs from the first phase
2: for $v = 2$ to $|\mathcal{M}|$ do
3:     if $\text{argmax}_{(a_i,m_j)\in\,\mathcal{F}}(r(a_i, v))$ is a unique pair $(a_x, m_y)$ then
4:         return $(a_x, m_y)$
5:     else
6:         $\mathcal{F}$ = the set of pairs returned by $\text{argmax}_{(a_i,m_j)\in\,\mathcal{F}}(r(a_i, v))$
7:     end if
8: end for
   /* program control reaches here only if no application, machine pair has been */
   /* selected in Lines 1 to 8 above. $\mathcal{F}$ is now the set of $(a_i, m_j)$ pairs from the last */
   /* execution of Line 6 */
9: if $\text{argmax}_{(a_i,m_j)\in\,\mathcal{F}}(T(a_i, 2))$ is a unique pair $(a_x, m_y)$ then
10:     return $(a_x, m_y)$
11: else
12:     arbitrarily select and return an application, machine pair from the set of pairs
        given by $\text{argmax}_{(a_i,m_j)\in\,\mathcal{F}}(T(a_i, 2))$
13: end if
```

Figure 11: Selecting the most critical application to map next given the set of (a_i, m_j) pairs from the first phase of MCTF.

The first phase does not make an application to machine assignment; it only establishes application-machine pairs (a_i, m_j) for all unmapped applications a_i. The second phase makes an application to machine assignment by selecting one of the (a_i, m_j) pairs produced by the first phase. It chooses the pair that maximizes $\Delta\mathbf{\Lambda}^*(a_i, m_j)$ over all first phase pairs. Ties are resolved arbitrarily.

```
1: while all applications are not mapped do
2:     for each unmapped application $a_i$ do
3:         find the machine $m_j$ such that $m_j = \text{argmax}_{m_k\in\,\mathcal{M}}(\Delta\mathbf{\Lambda}^*(a_i, m_k))$
4:         resolve ties arbitrarily
5:         if $\Delta\mathbf{\Lambda}^*(a_i, m_j) < 0$ then
6:             exit (this heuristic cannot find a resource allocation)
7:         end if
8:     end for
9:     from the $(a_i, m_j)$ pairs found above, select the pair(s) $(a_x, m_y)$ such that $(a_x, m_y) =$
       $\text{argmax}_{(a_i,m_j)\text{ pairs}}(\Delta\mathbf{\Lambda}^*(a_i, m_j))$
10:     resolve ties arbitrarily
11:     map $a_x$ on $m_y$
12: end while
```

Figure 12: The TPG heuristic.

7.3.4 Most Critical Path First Heuristic

The Most Critical Path First Heuristic (MCPF) heuristic explicitly considers the latency constraints of the paths in the system. It begins by ranking the paths in the order of the most "critical" path first (defined below). Then it uses a modified form of the MCTF heuristic to map applications on a path-by-path basis, iterating through the paths in a ranked order. The modified form of MCTF differs from MCTF in that the first iteration has been changed to be the same as the subsequent iterations.

The ranking procedure used by MCPF is now explained in detail. Let $\hat{\mathbf{\Lambda}}^{\mathrm{L}}(\mathcal{P}_k)$ be the value of $\Delta\mathbf{\Lambda}_k^{\mathrm{L}}$ assuming that each application a_i in $\mathcal{P}_k$ is mapped to the machine m_j where it has the smallest computation time, and that a_i can use 100% of m_j. Similarly for the communications between the consecutive applications in $\mathcal{P}_k$, it is assumed that the data transfer between the applications occurs over the highest speed communication link in the system, and that the link is 100% utilized by the data transfer. Note that the entire ranking procedure is done before any application is mapped.

The heuristic ranks the paths in an ordered list $\langle \mathcal{P}_1^{\mathrm{crit}}, \mathcal{P}_2^{\mathrm{crit}}, \cdots, \mathcal{P}_{|\mathcal{P}|}^{\mathrm{crit}} \rangle$ such that $\hat{\mathbf{\Lambda}}^{\mathrm{L}}(\mathcal{P}_x^{\mathrm{crit}}) \leq \hat{\mathbf{\Lambda}}^{\mathrm{L}}(\mathcal{P}_y^{\mathrm{crit}})$ if $x < y$. Once the ranking of the paths has been done, the MCPF heuristic uses MCTF to map each application in a path, starting from the highest ranked path first.

7.3.5 Duplex

For an arbitrary HC system, one is not expected to know if the system is more stringent with respect to latency constraints or throughput constraints. In that case, this research proposes running both MCTF and MCPF, and taking the better of the two mappings. The Duplex heuristic executes both MCTF and MCPF, and then chooses the resource allocation that gives a higher $\Delta\mathbf{\Lambda}$.

7.3.6 Other Heuristics

To compare the performance of the heuristics proposed in this research (MCTF, MCPF, and TPG), three other greedy heuristics were also implemented. These included: Two-Phase Greedy X (TPG-X) and two fast greedy heuristics. TPG-X is an implementation of the Max-min heuristic [29] for the environment discussed in this research. TPG-X is similar to the TPG heuristic except that in Line 9 of Figure 12, "argmax" is replaced with "argmin." The first fast greedy heuristic, denoted FGH-L, iterates through the unmapped applications in an arbitrary order, assigning an application a_i to the machine m_j such that (a) $\Delta\mathbf{\Lambda}^*(a_i, m_j) \geq 0$, and (b) $\Delta\mathbf{\Lambda}^{\mathrm{L}}$ is maximized (ties are resolved arbitrarily). The second fast greedy heuristic, FGH-T, is similar to FGH-L except that FGH-T attempts to maximize $\Delta\mathbf{\Lambda}^{\mathrm{T}}$. For a given application a_i, if FGH-L or FGH-T cannot find a machine m_j such that $\Delta\mathbf{\Lambda}^*(a_i, m_j) \geq 0$, then the heuristic fails.

7.3.7 An Upper Bound

An upper bound, UB, on the $\Delta\mathbf{\Lambda}$ value also is calculated for comparing the absolute performance of a given heuristic. The UB is equal to the $\Delta\mathbf{\Lambda}$ for a system where the following assumptions hold: (a) the communication times are zero for all applications, (b) each application a_i is mapped on the machine m_j where $\Delta\mathbf{\Lambda}_{ij}^{\mathrm{T}}$ is maximum over all machines, and (c) that each application can use 100% of the machine where it is mapped. These assumptions are, in general, not physically realistic.

8 Simulation Experiments and Results

In this study, several sets of simulation experiments were conducted to evaluate and compare the heuristics. Experiments were performed for different values of $|\mathcal{A}|$ and $|\mathcal{M}|$, and for different types of HC environments. For all experiments, it was assumed that an application could execute on any machine.

The following simplifying assumptions were made for performing the experiments. Let $\underline{n_s}$ be the total number of sensors. The estimated time to compute function $C_{ij}(\boldsymbol{\lambda})$ for application a_i on m_j was assumed to be of the form $\sum_{1\leq z\leq n_s} \underline{b_{ijz}}\lambda_z$, 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 given mean and given values of task heterogeneity and machine heterogeneity. The $T^{\mathrm{c}}_{ij}(\boldsymbol{\lambda})$ value would depend on the actual resource allocation as well as $C_{ij}(\boldsymbol{\lambda})$, and can be calculated using the computation model given in Subsection 7.2.1. Similarly, the $M_{ip}(\boldsymbol{\lambda})$ functions for the size of the message data sent from application a_i to a destination application a_p at a given load were similarly generated, except that machine heterogeneity was not involved. The communication time functions, $T^{\mathrm{n}}_{ip}(\boldsymbol{\lambda})$, would depend on the actual resource allocation as well as $M_{ip}(\boldsymbol{\lambda})$, and can be calculated using the communication model given in Subsection 7.2.2. For a given set of computation and communication time functions, the experimental set-up allowed the user to change the values of sensor output rates and end-to-end latency constraints so as to change the "tightness" of the throughput and latency constraints. The reader is directed to [6] for details.

An experiment is characterized by the set of system parameters (e.g., $|\mathcal{A}|$, $|\mathcal{M}|$, application and machine heterogeneities) it investigates. Each experiment was repeated 90 times to obtain good estimates of the mean and standard deviation of $\Delta\boldsymbol{\Lambda}$. Each repetition of a given experiment will be referred to as a trial. For each new trial, a DAG with $|\mathcal{A}|$ nodes was randomly regenerated, and the values of $C_{ij}(\boldsymbol{\lambda})$ and $M_{ip}(\boldsymbol{\lambda})$ were regenerated from their respective distributions.

Results from a typical set of experiments are shown in Figure 13. The first bar for each heuristic, titled "$\underline{\Delta\boldsymbol{\Lambda}^{\mathrm{N}}}$," shows the normalized $\Delta\boldsymbol{\Lambda}$ value averaged for all those trials in which the given heuristic successfully found a resource allocation. The normalized $\Delta\boldsymbol{\Lambda}$ for a given heuristic is equal to $\Delta\boldsymbol{\Lambda}$ for the resource allocation found by that heuristic divided by $\Delta\boldsymbol{\Lambda}$ for the upper bound defined in Subsection 7.3.7. The second bar, titled, "$\underline{\delta\boldsymbol{\lambda}^{\mathrm{N}}}$," shows the normalized $\Delta\boldsymbol{\Lambda}$ averaged only for those trials in which every heuristic successfully found a resource allocation. This figure also shows, in the third bar, the value of the failure rate for each heuristic. The failure rate or FR is the ratio of the number of trials in which the heuristic could not find a resource allocation to the total number of trials. The interval shown at the tops of the first two bars is the 95% confidence interval [30].

Figure 13 shows the relative performance of the heuristics for the given system parameters. In this figure, FGH-T and FGH-L are not shown because of their poor failure rate and $\Delta\boldsymbol{\Lambda}^{\mathrm{N}}$, respectively. It can be seen that the $\Delta\boldsymbol{\Lambda}^{\mathrm{N}}$ performance difference between MCTF and MCPF is statistically insignificant. The traditional Min-min and Max-min like heuristics, i.e., TPG and TPG-X, achieve $\Delta\boldsymbol{\Lambda}^{\mathrm{N}}$ values significantly lower than those for MCTF or MCPF (i.e., much poorer robustness). To make matters worse, the FR values for TPG and TPG-X are significantly higher than those for MCTF or MCPF. Even though Duplex's $\Delta\mathbf{\Lambda}^{\mathrm{N}}$ value is statistically no better than that of MCTF, its FR value, 12%, is about half that of MCTF (23%).

Additional experiments were performed for various other combinations of $|\mathcal{A}|$, $|\mathcal{M}|$, and tightness of QoS constraints, and the relative behavior of the heuristics was similar to that in Figure 13. Note that all communication times were set to zero in Figure 13 (but not in all experiments). Given the formulation of UB, it is expected that if the communication times are all zero in a given environment, then UB will be closer to the optimal value, and will make it easier to evaluate the performance of the heuristics with respect to the upper bound.

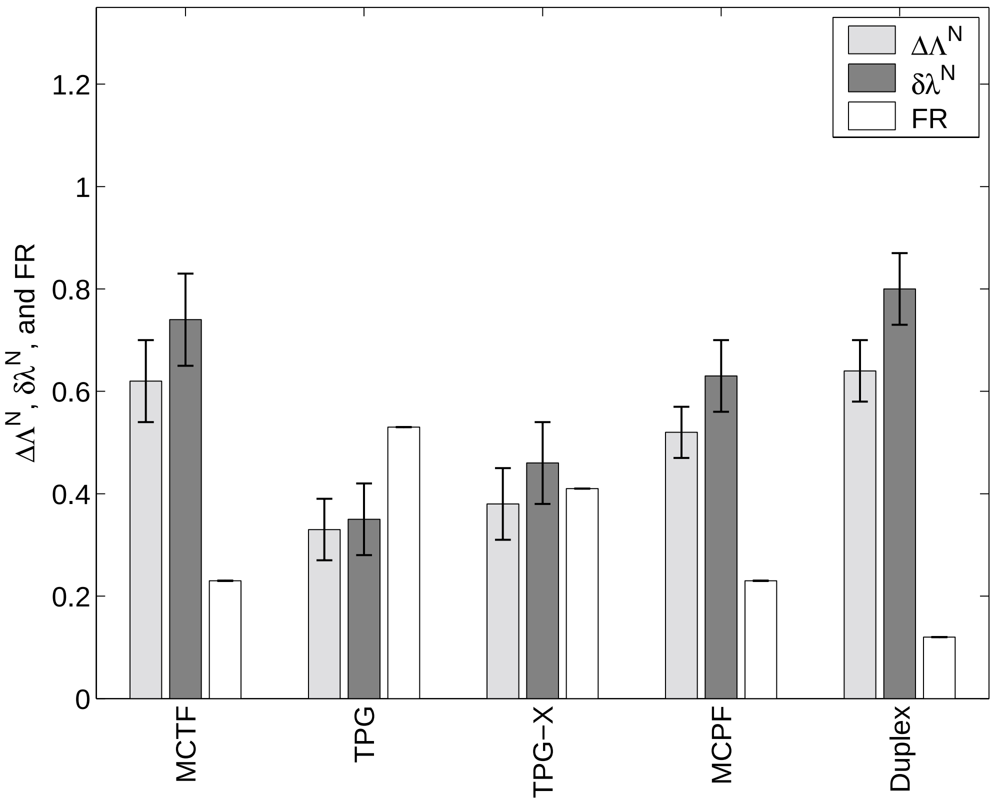


Figure 13: The relative performance of heuristics for a system where $|\mathcal{M}| = 6$, $|\mathcal{A}| = 50$. Number of sensors = number of actuators = 7. Task heterogeneity = machine heterogeneity = 0.7. All communication times were set to zero. A total of 90 trials were performed.

9 Related Work

Although a number of robustness measures have been studied in the literature (e.g., [10, 14, 17, 18, 19, 25, 31, 34, 38, 43]), those measures were developed for specific systems. The focus of the research in this chapter 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 chapter.

Given an allocation of a set of communicating applications to a set of machines, the work in [10] 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 [10], several theorems about the properties of robustness are proven. The robustness metric in [10] 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 [10] 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 [14], the authors address the issue of probabilistic guarantees for fault-tolerant real-time systems. As a first step towards 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 [17] 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. Like [10], the robustness metric in [17] was formulated for errors in the estimation of processing times, and was not intended for general use.

The study in [18] 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 study does not develop a robustness metric; instead, it implicitly uses slack to achieve robustness.

The Ballista project [19] 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). However, the robustness metric developed for that project is specific to software systems.

The research in [25] 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 re-execute, 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 [31], 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 [34] 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 [38] 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 [31], [38] does not explicitly state the perturbations under which the system is robust.

Our work is perhaps closest in philosophy to [43], 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.

Our earlier study in [2] is related to the one in Sections 7 and 8 chapter. However, the robustness measure we used in [2] makes a simplifying assumption about the way changes in $\boldsymbol{\lambda}$ can occur. Specifically, it is assumed that $\boldsymbol{\lambda}$ changes so that all components of $\boldsymbol{\lambda}$ increase in proportion to their initial values. That is, if the output from a given sensor increases by $x\%$, then the output from all sensors increases by $x\%$. Given this assumption, for any two sensors σ_p and σ_q, $(\lambda_p - \lambda_p^{\text{init}})/\lambda_p^{\text{init}} = (\lambda_q - \lambda_q^{\text{init}})/\lambda_q^{\text{init}} = \Delta\lambda$.

10 Future Work

There are many directions in which the robustness research presented in the paper can be extended. Examples include the following.

1. Deriving the boundary surfaces for different problem domains.

2. Incorporating multiple types of perturbation parameters (e.g., uncertainties in input sensor loads and uncertainties in estimated execution times). Challenges here are how

to define the collective impact to find each robust radius and how to state the combined bound on multiple perturbation parameters to maintain the promised performance.

3. Incorporating probabilistic information about uncertainties. In this case, a perturbation parameter can be represented as a vector of random variables. Then, one might have probabilistic information about random variables in the vector (e.g., probability density functions) or probabilistic information describing the relationship between different random variables in the vector or between different vectors (e.g., a set of correlation coefficients).

4. Determining when to use Euclidean distance versus other distance measures when calculating the collective impact of changes in the perturbation parameter elements.

11 Conclusions

This chapter has presented a mathematical description of a 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 computing 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 based on the primary performance measure (e.g., no constraint violation for HiPer-D environment and minimizing makespan for the cluster environments). It was shown that the robustness metric was more useful than approaches such as slack or load balancing.

Also, this chapter described several static resource allocation heuristics for one example distributed computing system. The focus was on designing a static heuristic that will (a) determine a maximally robust resource allocation, i.e., a resource allocation that maximizes the allowable increase in workload until a run-time reallocation of resources is required to avoid a QoS violation, and (b) have a very low failure rate. This study proposes a heuristic, called Duplex, that performs well with respect to the failure rate and the robustness towards unpredictable workload increases. Duplex was compared under a variety of simulated heterogeneous computing environments, and with a number of other heuristics taken from the literature. For all of the cases considered, Duplex gave the lowest failure rate, and a robustness value better than those of other evaluated heuristics. Duplex is, therefore, very desirable for systems where low failure rates can be a critical requirement and where unpredictable circumstances can lead to unknown increases in the system workload.

Acknowledgments: The authors thank Vladimir Shestak for his valuable contributions.

References

[1] S. Ali, T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "Characterizing resource allocation heuristics for heterogeneous computing systems," *Parallel, Distributed, and Pervasive Computing,*

A. R. Hurson, ed., Vol. 63 of *Advances in Computers*, Elsevier Academic Press, San Diego, CA, 2005, pp. 91–128.

[2] 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," *2002 International Conference on Parallel and Distributed Processing Techniques and Applications, Vol. II*, June 2002, pp. 519–530.

[3] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Measuring the robustness of a resource allocation," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 7, July 2004, pp. 630–641.

[4] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim, "Robust resource allocation for sensor-actuator distributed computing systems," *2004 International Conference on Parallel Processing (ICPP 2004)*, Aug. 2004, pp. 178–185.

[5] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Sedigh-Ali, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering*, Vol. 3, No. 3, Nov. 2000, pp. 195–207, invited.

[6] S. Ali, *Robust Resource Allocation in Dynamic Distributed Heterogeneous Computing Systems*. PhD thesis, School of Electrical and Computer Engineering, Purdue University, Aug. 2003.

[7] H. Barada, S. M. Sait, and N. Baig, "Task matching and scheduling in heterogeneous systems using simulated evolution," *10th IEEE Heterogeneous Computing Workshop (HCW 2001) in the proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)*, Apr. 2001.

[8] I. Banicescu and V. Velusamy, "Performance of scheduling scientific applications with adaptive weighted factoring," *10th IEEE Heterogeneous Computing Workshop (HCW 2001) in the proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)*, Apr. 2001.

[9] 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.

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

[11] S. Boyd and L. Vandenberghe, *Convex Optimization,* available at `http://www.stanford.edu/class/ee364/index.html`.

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

[13] T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Heterogeneous computing: Goals, methods, and open problems (invited keynote presentation for the 2001 International Multiconference that included PDPTA 2001)," *2001 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001),* Vol. I, pp. 1–12, June 2001.

[14] 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, UK, 1997.

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

[16] E. G. Coffman, Jr., ed., *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons, New York, NY, 1976.

[17] R. L. Daniels and J. E. Carrillo, "β-Robust scheduling for single-machine systems with uncertain processing times," *IIE Transactions*, Vol. 29, No. 11, 1997, pp. 977–985.

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

[19] J. DeVale and P. Koopman, "Robust software – no more excuses," *IEEE International Conference on Dependable Systems and Networks (DSN 2002)*, June 2002, pp. 145–154.

[20] M. M. Eshaghian, ed., *Heterogeneous Computing*, Artech House, Norwood, MA, 1996.

[21] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transaction on Software Engineering*, Vol. SE-15, No. 11, Nov. 1989, pp. 1427–1436.

[22] I. Foster and C. Kesselman, eds., *The Grid 2: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann, San Fransisco, CA, 2004.

[23] I. Foster, *Designing and Building Parallel Programs*, Addison-Wesley, Reading, MA, 1995.

[24] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 13–17.

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

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

[27] E. Hart, P. M. Ross, and J. Nelson, "Producing robust schedules via an artificial immune system," *1998 International Conference on Evolutionary Computing*, May 1998, pp. 464–469.

[28] 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.

[29] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, Vol. 24, No. 2, Apr. 1977, pp. 280–289.

[30] R. Jain, *The Art of Computer Systems Performance Analysis*, John Wiley & Sons, Inc., New York, NY, 1991.

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

[32] E. Jen, "Stable or robust? What is the difference?," *Santa Fe Institute Working Paper No. 02-12-069*, 2002.

[33] A. Khokhar, V. K. Prasanna, M. Shaaban, and C. L. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, Vol. 26, No. 6, June 1993, pp. 18–27.

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

[35] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, Vol. 59, No. 2, Nov. 1999, pp. 107–131.

[36] Z. Michalewicz and D. B. Fogel, *How to Solve It: Modern Heuristics*, Springer-Verlag, New York, NY, 2000.

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

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

[39] V. Shestak, H. J. Siegel, A. A. Maciejewski, and S. Ali, "Robust resource allocations in parallel computing systems: Model and heuristics," *The 2005 IEEE International Symposium on Parallel Architectures, Algorithms, and Networks*, Dec. 2005.

[40] S. Shivle, P. Sugavanam, H. J. Siegel, A. A. Maciejewski, T. Banka, K. Chindam, S. Dussinger, A. Kutruff, P. Penumarthy, P. Pichumani, P. Satyasekaran, D. Sendek, J. Smith, J. Sousa, J. Sridharan, and J. Velazco, "Mapping subtasks with multiple versions on an ad hoc grid," *Parallel Computing,* Special Issue on Heterogeneous Computing, Vol. 31, No. 7, jul 2005, pp. 671–690.

[41] V. Shestak, H. J. Siegel, A. A. Maciejewski, and S. Ali, "The robustness of resource allocations in parallel and distributed computing systems," *19th IEEE International Conference on Architecture of Computing Systems: System Aspects in Organic Computing (ARCS 2006)*, Mar. 2006.

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

[43] 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., Kluwer Academic Publishers, Norwell, MA, 1998, pp. 72–108.

[44] P. Sugavanam, H. J. Siegel, A. A. Maciejewski, M. Oltikar, A. Mehta, R. Pichel, A. Horiuchi, V. Shestak, M. Al-Otaibi, Y. Krishnamurthy, S. Ali, J. Zhang, M. Aydin, P. Lee, K. Guru, M. Raskey, and A. Pippin, “Robust static allocation of resources for independent tasks under makespan and dollar cost constraints,” *Journal of Parallel and Distributed Computing*, accepted, to appear.

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

[46] M.-Y. Wu, W. Shu, and H. Zhang, “Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems,” *9th IEEE Heterogeneous Computing Workshop (HCW 2000)*, May 2000, pp. 375–385.