DISSERTATION

Robust Resource Allocation in Heterogeneous Parallel and Distributed Computing Systems

Submitted by

James T. Smith II

Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2008

UMI Number: 3332752

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform 3332752

Copyright 2008 by ProQuest LLC.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 E. Eisenhower Parkway
PO Box 1346
Ann Arbor, MI 48106-1346

Copyright by James T. Smith II 2008

All Rights Reserved

COLORADO STATE UNIVERSITY

June 27, 2008

WE HEREBY RECOMMEND THAT THE **DISSERTATION** PREPARED UNDER OUR SUPERVISION BY **JAMES T. SMITH II** ENTITLED **ROBUST RESOURCE ALLOCATION IN HETEROGENEOUS PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS** BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Prof. E. K. P. Chong

Prof. P. Burns
(Mechanical Engineering)

Prof. H. J. Siegel
Adviser

Prof. A. A. Maciejewski
Co-Adviser

Prof. A. A. Maciejewski
Department Head/Director

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

To my wife Cary and my children Oliver, Paxon, and Emlen for their patience and support.

To my parents for their constant encouragement.

ACKNOWLEDGEMENTS

I would like to thank my advisors Prof. Siegel and Prof. Maciejewski for their guidance and help through this process. Special thanks to Prof. Chong and Prof. Burns. Thanks to Vladimir Shestak, Samee Khan, Luis Briceño, Ricky Kwok, Jerry Potter, Chris Klumph, and Kody Willman for their valuable comments on various portions of this work. This research was supported by the NSF under Grant CNS-0615170 and by the Colorado State University George T. Abell Endowment.

ABSTRACT OF DISSERTATION

Robust Resource Allocation in Heterogeneous

Parallel and Distributed Computing Systems

In a heterogeneous distributed computing environment, it is often advantageous to allocate system resources in a manner that optimizes a given system performance measure. However, this optimization is often dependent on system parameters whose values are subject to uncertainty. Thus, an important research problem arises when system resources must be allocated given uncertainty in system parameters. Robustness can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed. In this research, we define mathematical models of robustness in both static and dynamic stochastic environments. In addition, we model dynamic environments where estimates of system parameter values are provided as point estimates where these estimates are known to deviate substantially from their actual values.

The main contributions of this research are (1) mathematical models of robustness suitable for dynamic environments based on single estimates of system parameters (2) a mathematical model of robustness applicable to environments where the uncertainty in system parameters can be modeled stochastically, (3) a demonstration of the use of this metric to design resource allocation heuristics in a static environment, (4) a mathematical model of robustness in a stochastic dynamic environment, (5) we demonstrate the utility of this dynamic robustness metric through the design of resource allocation heuristics, (6) the derivation of a robustness metric for evaluating resource allocation decisions in an overlay network along with a near optimal resource allocation technique suitable to this environment.

James T. Smith II
Electrical and Computer Engineering
Colorado State University
Fort Collins, CO 80523
Summer 2008

CHAPTER 1

INTRODUCTION

In a heterogeneous distributed computing environment, it is often advantageous to allocate system resources in a manner that optimizes a given system performance measure. However, this optimization is often dependent on system parameters that are subject to uncertainty. Thus, an important research problem arises where system resources must be allocated given uncertainty in system parameters. Robustness can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed. Existing research in this area has focused on static resource allocation environments where no prior information is available for characterizing the uncertainty in system parameters.

The main contributions of this research are (1) a mathematical derivation of robustness suitable for a dynamic environment based on point estimates of system parameters (2) a mathematical definition of robustness applicable to environments where the uncertainty in system parameters can be modeled stochastically, (3) a demonstration of the use of this metric to design resource allocation heuristics in a static environment, (4) a mathematical definition of robustness in a stochastic dynamic environment, (5) a demonstration of the use of this dynamic robustness metric to design resource allocation heuristics suitable for a given heterogeneous computing system, (6) the derivation of a robustness metric for resource allocations in an overlay network along with a near optimal resource allocation technique for this environment.

Chapter 2 provides an overview of robust resource allocation in heterogeneous parallel and distributed computing systems. In Chapter 3, we derive a robustness metric for a dynamic environment where each task is subject to a hard deadline. We derive a robustness metric for a dynamic resource allocation environment in Chapter 4 where all tasks are required to be completed by a common deadline. In Chapter 5, we define a methodology for quantifying the robustness of resource allocations in environments where the uncertainty

in system parameters can be modeled stochastically. We analyze the effectiveness of our approach through direct comparison with prior work in this area. We demonstrate a design methodology in Chapter 6 for utilizing this robustness metric to produce effective resource allocation heuristics in a static resource allocation environment where we wish to minimize the time required to complete all tasks. In Chapter 7, we derive a methodology for maximizing the robustness of resource allocations where all tasks must be completed by a fixed deadline.

In chapter 8, we derive a robustness metric suitable for a dynamic stochastic environment. We derive a mathematical formulation of robustness in such a stochastic dynamic environment and analyze the effectiveness of the metric for evaluating a select group of resource allocation heuristics. We present an alternate formulation of robustness in a stochastic dynamic environment in Chapter 9 that can be used to aid resource allocation decision making. We design two novel resource allocation techniques that utilize our robustness metric during resource allocation.

In Chapter 10, we consider the optimization of resource allocations in an overlay network where the arrival rate of packets to the network is not known in advance. We derive a robust decentralized methodology for resource allocation in such an environment.

CHAPTER 2

ROBUST RESOURCE ALLOCATION IN HETEROGENEOUS PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS

2.1 Introduction

In parallel and distributed computing multiple computers are collectively utilized to simultaneously process a set of tasks to improve performance over that of a single processor [26]. Often, such computing systems are constructed from a heterogeneous mix of machines that may differ in their capabilities, e.g., available memory, number of floating point units, clock speed, and operating system. In a heterogeneous computing system, the execution time of a task may differ depending on which computer executes the task. Often, task resource requirements lead to inconsistent performance differences between heterogeneous machines. That is, machine 1 being faster than machine 2 on some task A does not imply that machine 1 is uniformly faster on all tasks.

Resource allocation in heterogeneous parallel and distributed computing is the process of assigning tasks to computers for execution such that some performance objective is optimized. For example, a common objective in resource allocation is to minimize the total time required to complete a set of tasks to be executed. It has been shown that resource allocation is an NP-hard problem (e.g., [50]), i.e., an optimal solution cannot be found in reasonable time for problems of realistic size. Therefore, the task of resource allocation is often addressed heuristically. A resource allocation heuristic generates task to machine assignments that attempt to optimize the identified performance objective. The design of heuristics for resource allocation is an active area of research, e.g., [25,36]. In this chapter, we will evaluate the robustness of resource allocations in both static and dynamic

---

The research presented in this chapter has been accepted to appear in [97].

environments. In a static environment, the entire collection of tasks to be allocated is known in advance, prior to the start of allocation. In contrast, in a dynamic environment, the set of tasks to be executed is not known in advance and tasks are assigned as they arrive to the system.

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

In the next section, we describe the formal FePIA procedure for deriving a robustness metric for any given system. In Section 2.3, we provide an example robustness metric derivation for environments where uncertainties are due to system parameters whose values are only estimates. We use the steps of the FePIA procedure in Section 2.4 to define a model of robustness in a stochastic environment where uncertainties are modeled as random variables. We conclude with a brief discussion of the open problems in robust resource allocation.

2.2 Determining a Robustness Metric

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

(4) analyze the system to quantify robustness.

To illustrate the intuition behind the steps of the FePIA procedure, we will use the following simple resource allocation example throughout our discussion. In this example, a set of tasks is to be assigned to a heterogeneous collection of machines such that the finishing time of the last to finish machine, i.e., the total time required to complete all of the tasks (often referred to as makespan), is minimized. An estimate of the execution time of each task on each machine is available, and the resource allocation must take into account that there are unknown errors in these estimates (e.g., their actual values may be data dependent). The following is a summary of the four steps of the FePIA procedure, first presented in [3].

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

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

errors in task execution time estimates.

Step 3: Identify the impact of perturbation parameters (identified in step 2) on the performance features of the system (identified in step 1). With respect to our robustness requirement from step 1, the actual value of our performance feature must be within the identified acceptable level of variation. For the makespan example, the sum of the *actual* execution times for all tasks assigned to any given machine determines the *actual* finishing time of that machine. Thus, the actual finishing time of the last to finish machine, i.e., the actual makespan, must be less than or equal to τ. Differences between the *estimated* task execution times and their actual values will directly impact the ability of the system to meet the robustness requirement established in step 1.

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

2.3 Deterministic Models of Robustness

2.3.1 Introduction

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

2.3.2 Example Static Environment

2.3.2.1 Deriving a Robustness Metric

In this environment, similar to the makespan example of the previous section, the goal of the resource allocation is to assign $\underline{T}$ tasks to $\underline{M}$ machines such that the robustness of the system is maximized [100]. Because this is a static environment, the T tasks to be assigned

are all known in advance. Further, we assume that the estimated time to compute (ETC) each task on each machine has been provided (determined by experimental or analytical techniques [64]). We also assume that the M machines are heterogeneous, i.e., each ETC value for a given task on the M machines may be different. Using the FePIA procedure, we can quantitatively define a measure of robustness for this example system as follows.

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

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

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

Step 4: Finally, we conduct an analysis to determine exactly how robust the system is under a specific resource allocation. The robustness radius of a performance feature,

denoted $r_\mu(F_j(C^{\text{est}}, \mu), C)$, is defined as the smallest collective increase in system parameters that would lead to a violation of the robustness requirement for that performance feature. That is, for a given machine j, we would like to know the smallest collective increase in the execution times of tasks assigned to that machine that would result in $F_j(C, \mu) > \tau$. Quantitatively, if the Euclidean distance between the vector of actual computation times and the vector of estimated computation times for tasks assigned to machine j is no larger than $r_\mu(F_j(C^{\text{est}}, \mu), C)$, then the finishing time of machine j will be less than the makespan constraint τ. Because the finishing time of a machine is the sum of the execution times of all tasks assigned to that machine, the makespan contsraint can be represented as a hyperplane in a multi-dimensional space whose axes are defined for each machine in terms of the tasks assigned it. That is, each perturbation parameter provides a single dimension along which the robustness radius can vary. For example, in Figure 1, the two tasks c_1 and c_2 have been assigned to machine j and provide the axes for this geometric analysis. The radius $r_\mu(F_j(C^{\text{est}}, \mu), C)$, interpreted geometrically, is the shortest distance from C^{est} to the hyperplane given by $F_j(C^{\text{est}}, \mu) = \tau$. Thus, as long as the estimation error contained in C^{est} does not exceed $r_\mu(F_j(C^{\text{est}}, \mu))$ then the finishing time of machine j will not exceed τ. In Figure 1, the robustness requirement is plotted as a solid gray line—highlighting the boundary between robust performance and non-robust performance. The estimated execution times of c_1 and c_2 are plotted together in the figure as the estimated value of the perturbation parameters. Figure 1 demonstrates the robustness radius for this example as the shortest distance from the point estimate of the perturbation parameters to the hyperplane defined by the robustness requirement, i.e., a line in two dimensions. The general calculation of $r_\mu(F_j(C^{\text{est}}, \mu), C)$ can be expressed using the point-to-plane distance formula as follows,

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

Finally, the robustness metric can be expressed in terms of the set of robustness radii, where the metric is equal to the smallest of the robustness radii for the set of performance features

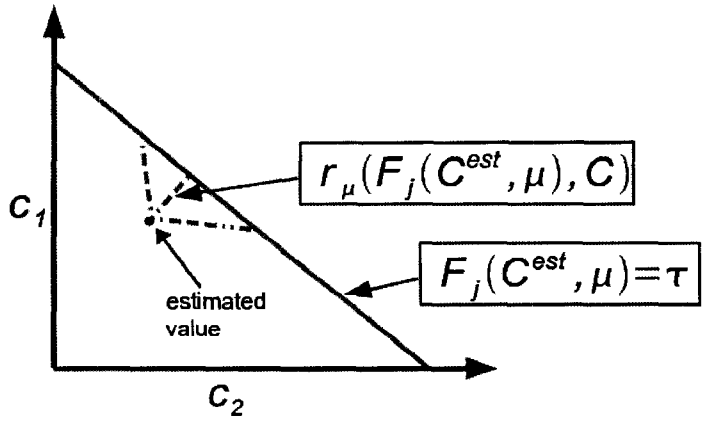


Figure 1: An example geometric analysis of the robustness radius for a given machine. In the example, resource allocation μ includes the assignment of tasks c_1 and c_2 to a single machine j. The robustness requirement for the finishing time of this machine, i.e., $F_j(C^{\text{est}}, \mu) = \tau$, can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

ϕ, denoted $\underline{\rho_\mu}$. Mathematically, this is expressed as follows,

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

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

2.3.2.2 Using Static Robustness

The robustness metric can be used directly to compare resource allocations for their ability to deliver on promised performance, as in [3]. Alternatively, the robustness metric can be used during resource allocation to improve the overall robustness of the resulting allocation. That is, using the robustness metric of Subsection 2.3.2.1, we can define a resource allocation heuristic that attempts to maximize the robustness of the resource allocations that it produces. In this subsection, we give an example resource allocation heuristic that

attempts to maximize robustness.

The max-max heuristic, presented in [100], uses a measure of robustness during allocation to maximize robustness. Max-max starts with an initial set of tasks to be assigned to machines within the system. For each task, we consider what the robustness radius for each machine would be if that task were assigned to it and select the machine that provides the largest overall robustness radius. From this set of task-machine pairs, max-max selects the task-machine pair that provides the overall largest robustness radius. The selected task is assigned to its selected machine, removed from the set of tasks to be assigned, and the machine's completion time is updated accordingly. Max-max continues in this way until all tasks have been assigned to machines in the heterogeneous system. Note that when evaluating $r_\mu(F_j(C^{\text{est}}, \mu), C)$, as in Equation 1, during resource allocation we replace the complete machine finishing time $F_j(C^{\text{est}})$ with the finishing time of the machine given the set of tasks that have previously been assigned to this machine plus the task under consideration. This value can be thought of as an intermediate robustness radius determined by the partial allocation when it is calculated.

This heuristic was evaluated in [100] alongside several other resource allocation techniques. By performing resource allocation in this way, the resulting allocation is able to tolerate a larger variation in task execution times than many commonly used techniques, as demonstrated in [100]. The paper also demonstrates the utility of this approach through comparison with some other techniques for resource allocation.

2.3.3 Example Dynamic Environment

2.3.3.1 Deriving a Robustness Metric

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

the actual execution times may vary due to a dependence on the characteristics of the input data that are not known until execution time.

Tasks in a dynamic environment can be assigned in either immediate mode or batch mode. In immediate mode, tasks are assigned immediately as they arrive. In batch mode, tasks are instead collected as they arrive and assigned as a batch [66, 72]. The robustness metric developed in this subsection is appropriate for use in heuristics that operate in either immediate mode or batch mode. In the next subsection, we will give an example of an immediate mode heuristic developed for this environment.

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

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

According to Section 2.1, we can intuitively define robustness by considering answers to the three questions of that section. What behavior makes the system robust? To be robust, the finishing time of each machine at each mapping event should be limited in variation. What uncertainties is the system robust against? In this environment, unknown estimation errors in the ETC values may cause machine finishing times to increase unpredictably. Quantitatively, exactly how robust is the system? To answer this question, we have to determine exactly how much variation in task execution times can be tolerated while still

ensuring that the finishing time of each machine at each mapping event remains within a allowed range. We address these points more precisely using the FePIA procedure to derive a robustness metric for this environment.

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

$$\beta(t) = \max_{\forall j \in M} \left(F_j(t) \right). \tag{4}$$

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

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

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

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

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

In the static environment of the previous subsection, the number of robustness radii was equivalent to the number of machines in the computing system. However, in this environment we need to measure the robustness radius of each machine at each point in

time where our information about the robustness radius may change. Let $\underline{\rho_\mu(t)}$ be the robustness of the resource allocation as measured at time t and found as follows,

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

Thus, because the mix of tasks pending execution changes whenever a task arrives and all tasks must complete, there will be $2 \times T \times M$ robustness radii to be considered. The minimum over all of these robustness radii will provide the robustness metric, denoted $\underline{\rho_\mu}$, for the resource allocation. Let E be the set of all times when mapping events occur in the system during a resource allocation. Mathematically, ρ_μ can be expressed as follows,

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

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

2.3.3.2 Using Dynamic Robustness

This subsection uses the example problem formulation of the previous subsection (presented in [72]) and focuses on generating a dynamic resource allocation for a set of dynamically arriving, independent tasks. The resource allocation is expected to minimize $\beta(t)$, while still being able to tolerate a quantifiable amount of variation in the ETC values for the assigned tasks. Therefore, the goal of heuristics in this environment is to assign tasks to machines such that βt is minimized at each mapping event while still maintaining a specified level of robustness, e.g., $\rho_\mu(t) \geq \alpha$ at each mapping event. The following is an example of an immediate mode heuristic, known as feasible k-percent best, that successfully addresses these competing concerns by iteratively reducing the set of machines under consideration until a "best" machine has been selected.

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

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

2.4 Stochastic Models of Robustness

2.4.1 Introduction

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

that characterizes this uncertainty beyond a simple point estimate, as used in the previous section.

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

2.4.2 Stochastic Robustness in a Static Environment

2.4.2.1 Deriving a Robustness Metric

In this sample environment, we are concerned with allocating a set of T tasks to M heterogeneous machines where we are concerned with system makespan as the performance metric. Below, we use the FePIA procedure to derive a robustness metric for this environment.

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

Step 2: Uncertainty in this system arises because the exact execution time for each task is unknown. We can model the execution time of each task i on each machine j as a random variable [104], denoted $\underline{\eta_{ij}}$.

Step 3: The finishing time of each machine in a stochastic environment is calculated as the sum of the execution time random variables for each task assigned to that machine [86]. Let $\underline{n_j}$ be the count of the number of tasks assigned to machine j. The finishing time of

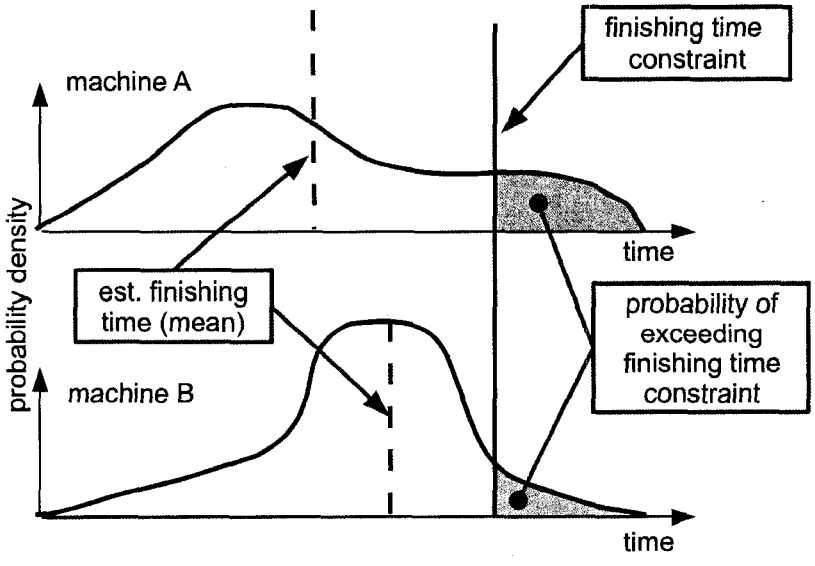


Figure 2: An example system where we are to assign task 3 to either machine A or machine B and we would like task 3 to complete prior to the plotted completion time constraint. Presented are the task completion time probability density functions for task 3 on both machine A and machine B. Using the point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

machine j, referred to as a local performance characteristic ψ_j, can be expressed as follows,

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

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

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

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

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

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

$$\theta = \prod_{\forall j} \left(\mathbb{P}[\psi_j \leq \beta^{max}] \right). \tag{10}$$

If the execution times η_{ij} for tasks assigned to machine j are mutually independent, then the summation of Equation 8 can be computed using an $(n_j - 1)$-fold convolution of the corresponding pmfs [63, 86].

2.4.2.2 *Using Static Robustness*

Two ways of using the static stochastic robustness metric are apparent by inspecting the parameters of Equation 10. The two parameters β^{max} and θ can alternately be either optimized or specified by the user. For example, the user could specify a β^{max} value as a constraint and employ a heuristic to attempt to maximize the robustness (θ) of the resulting resource allocation. That is, in this case, the user is interested in a resource allocation that has the highest probability to complete all of the tasks by β^{max}. Maximizing the robustness of a resource allocation given a fixed β^{max} requires minimizing the ψ_j values, thus, maximizing the probability that each machine will finish before β^{max}.

For some systems, it may be unclear how to select an appropriate β^{max} value. Thus, we can instead define a minimum acceptable θ value, denoted ω, and attempt to minimize β^{max} such that the probability that all tasks complete by β^{max} is at least ω. In this subsection, we consider the case where the minimum acceptable robustness value ω is specified by the user and we want to minimize β^{max} such that $\theta \geq \omega$. In [87], the period minimization routine (PMR) was introduced to iterate through the possible β^{max} values for a *given resource allocation* to find the smallest β^{max} value, provided by that allocation, such that $\theta \geq \omega$.

To demonstrate the use of the stochastic robustness metric in a resource allocation, we introduce a static stochastic environment where a set of machines periodically receive data sets to be processed by a collection of n tasks [87]. Each data set must be processed by all of the tasks before the next data set arrives. The execution times for each of the n tasks is assumed to be inherently data dependent, thus, the exact execution time for each task is unknown prior to its execution. However, we are provided a pmf describing the probabilities of task execution times for each machine.

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

The stochastic robustness metric for this system is the probability that the actual

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

We can use this formulation of robustness along with the PMR procedure to design a two-phase greedy heuristic for resource allocation in this system [87]. The two-phase greedy heuristic first initializes the set of tasks to be executed to the entire set of tasks that are available. While there are still tasks to execute, the heuristic uses two phases to find the next task assignment. In the first phase, the heuristic determines a machine assignment for each unassigned task that minimizes β^{max} (ignoring all other unassigned tasks), where we use the PMR procedure to determine the smallest β^{max} for each possible allocation such that the robustness constraint (ω) is still satisfied. In the second phase, the heuristic selects the task machine pair (found in the first phase) that provides the smallest overall β^{max}. The selected task is then allocated to its chosen machine and removed from the set of tasks to be assigned. The heuristic continues in this way until all tasks have been allocated.

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

2.5 Open Problems

The open problems in robust resource allocation are directly related to the long term goals of heterogeneous computing research [56]. The principal goal in heterogeneous computing (HC) research is to develop software environments that automatically assign and execute applications, where applications are expressed in a machine independent, high-level language. Developing such environments will facilitate the use of heterogeneous computing by (1) increasing software portability (i.e., programmers need not be concerned with the machine details of the heterogeneous environment) and (2) increasing the possibility of deriving better task machine assignments than users themselves derive using *ad-hoc* methods.

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

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

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

In stage 3, task profiling data and analytical benchmarking results are combined to create execution time estimates for each task and subtask on each machine in the suite. These results, along with initial loading and "status" of each machine in the suite, are used to generate machine assignments, based on a chosen optimization criteria, for each task and each subtask to be executed. Hierarchical scheduling techniques are of interest to allow the development of very large HC environments, e.g., grids [44]. In some environments,

the scale of the distributed system requires that task assignment be done in a distributed fashion, as opposed to a centralized resource allocation.

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

To realize this automatic heterogeneous computing environment requires further research in many areas. For example in stage 2, machine independent languages with user-specified directives are needed to (1) allow compilation of applications into efficient code suitable for any machine in an HC system, (2) aid in decomposing applications into tasks and subtasks, and (3) facilitate the determination of task and subtask computational requirements.

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

Another area of research related to stage 2 involves modeling uncertainty in perturbation parameters for robust resource allocation. For example, in a stochastic environment, methods are needed for leveraging experiential data to model uncertainty in perturbation parameters [51]. This is important because prior work in stochastic resource allocation environments has assumed that these models are available, i.e., in the form of probability mass functions. Further, once a pmf has been established for a perturbation parameter, methods are needed for updating the existing pmf with new experiential data. Updating pmfs with

the most current information is important in a dynamic environment because it enables the model to track changes in perturbation parameter distributions over time. In addition, pmfs based on experiential data may provide only an approximation of the true distribution of perturbation parameter values and methods are needed for determining the impact of estimation error in pmfs on robustness calculations. That is, how to make resource allocation decisions robust with respect to estimation errors in perturbation parameter pmfs is an open problem.

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

Current research in robust resource allocation, related to stage 3, is investigating combining multiple types of perturbation parameters into a single robustness metric, e.g., combining machine failure probabilities and task execution time uncertainty into a single metric. Combining multiple perturbation parameters into a single measure of robustness will extend the applicability of robust resource allocation into problem domains where these uncertainties occur simultaneously.

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

execution.

2.6 Summary

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

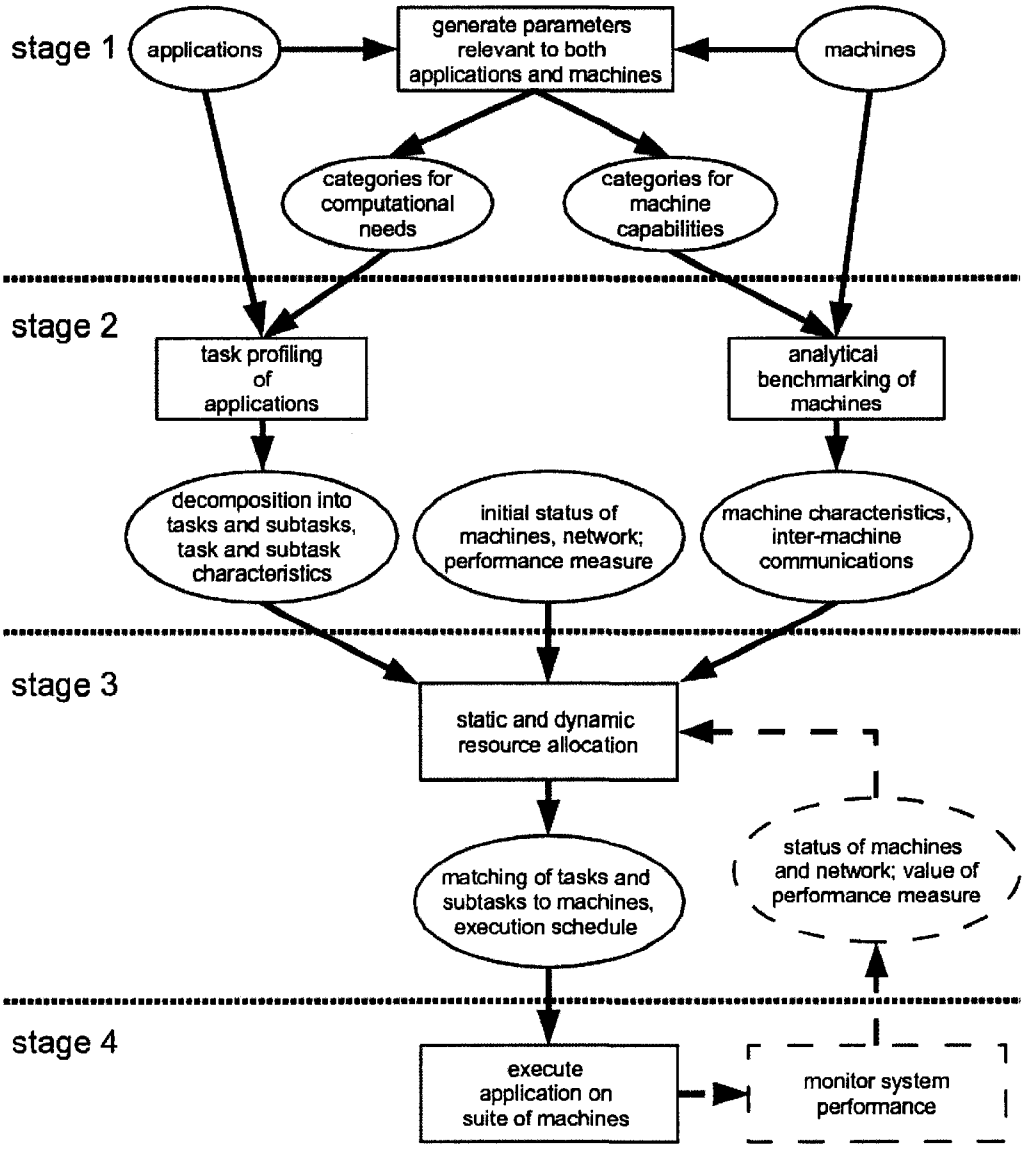


Figure 3: A model of required support for utilizing heterogeneous computing environments. Ovals indicate information and rectangles actions. The dashed lines represent the components needed to perform a dynamic resource allocation.

CHAPTER 3

ROBUST RESOURCE ALLOCATION IN A CLUSTER BASED IMAGING SYSTEM

3.1 Introduction

Recently there has been an increased demand for imaging systems in support of high-speed color digital printing. Increases in print speeds and resolution have necessitated a significant increase in the performance of imaging systems in support of digital printing systems. This required increase in performance can be achieved through an effective parallel execution of image processing applications in a distributed computing environment. In this paper, we present a mathematical model of a distributed raster imaging system, where the output of the system must be presented to a raster based display at fixed regular time intervals, effectively establishing a hard deadline for the completion of each output image. This mathematical model is used as the basis for the design of a resource allocation heuristic applicable to this distributed computing environment. We extend the use of our model by deriving a robustness metric appropriate to this environment. This robustness metric is used within our presented resource allocation heuristic as an alternate optimization criterion for the heuristic.

In this system, an input stream of data, described using a high level language known as a page description language (pdl), e.g., postscript or the portable document format (pdf), arrives at an imaging system for rasterization [47]. Rasterization of pdl images converts the images from a pdl description to a bitmap. Requests for rasterization of pdl images are processed by a dedicated cluster of workstations, where individual pdl image requests, referred to as sheetsides, are distributed to the heterogeneous cluster by a centralized image

The research presented in this chapter was jointly conducted with my colleagues Vladimir Shestak and Prasana Sugavanum [95].

dispatcher. The collection of sheetside requests together describe an image stream that is displayed on a raster based device, e.g., a printer or computer monitor. The frequency of requests and the magnitude of the data required to describe each request pose a considerable challenge for even modern workstations. Input streams in this environment routinely consist of over 100,000 images, where each image typically requires 10–100 megabytes of storage and successive image deadlines are on the order of a tenth of a second apart.

For the studied environment, the images that comprise the input datastream are required to be displayed *in order* on the output device. That is, each pdl image has a unique number assigning its place in the overall stream of images and will be requested by the raster display in that order. In addition, the system has a finite amount of storage capacity (distributed evenly across the cluster of workstations) in which to store rasterized images. The bitmaps to be displayed are retrieved at a regular interval directly from the workstation output buffers by the display device. Bitmaps are displayed for a fixed time interval, thus, the display time of the first bitmap establishes a hard deadline for each subsequent bitmap. Missing a deadline for a required bitmap results in an interruption of service that is unacceptable.

The studied rasterization system has some additional special requirements that complicate the task of assigning the stream of incoming pdl images to available workstations. The computation required to convert a pdl image to a bitmap depends on the content of the pdl file. The system only has an estimate of the time required to rasterize each incoming pdl image on each type of processor and this estimate may differ substantially from the actual time required for rasterization. Many of the system design decisions are motivated by an attempt to mitigate the impact of this uncertainty.

Second, the overall system has finite input and output storage capacity, thus, there is a limit on the number of pdl images that can be buffered in the system, both as input pdl images and as output bitmaps. Finally, pdl images continue to arrive for rasterization while others are being rasterized, i.e., the resource allocation must be produced dynamically [66]. The general problem of assigning tasks to workstations in a dynamic environment has been shown to be NP-complete (e.g., [32, 42, 50]). Consequently, the design of heuristics for

dynamic resource allocation is an active area of research [12, 44, 66, 72, 93, 107].

The concept of robustness of a resource allocation was introduced in [3, 89], and later efforts applied the concept to resource management [5, 72, 88, 89, 93, 100]. The mathematical model presented in this work builds upon principles addressed in our earlier work on robustness. In analyzing this image processing system, we identified that rasterization times are a source of uncertainty in the system and that the arrival ordering of sheetsides is not known *a priori*. This uncertainty can impact the system by causing sheetsides to miss their deadline, resulting in an interruption of service that is unacceptable in this system. Clearly, the area of robust operation within this system exists where bitmaps are always available in advance of their deadlines.

The primary contributions of this work are: (1) a mathematical model of a distributed raster image processing system, (2) the derivation of a robustness metric for a dynamic distributed computing system with hard deadlines for task completions, and (3) the design of the resource allocation heuristics suitable for this type of system. We clearly demonstrate the superiority of our heuristic technique (using two different optimization criteria) over a technique commonly used in this type of environment.

The details of the system model that motivated this research are given in the next Section. This system model is used to design a mathematical model of rasterization completion times presented in Section 3.3. Section 3.4 describes a new resource allocation heuristic that incorporates this mathematical model for rasterization completion times. We present a discussion of the performance objective for this initial heuristic in Section 3.5. This performance metric motivated the derivation of the robustness metric in Section 3.6. The details of the simulation setup are described in Section 3.7 and the results of the heuristics are presented in Section 3.8. A sampling of related work is in Section 3.9 and Section 3.10 concludes the paper.

3.2 System Model

Figure 4 is a conceptual drawing of the system that motivated this research. In this environment, two display devices combine to provide a high-speed digital continuous-form, color

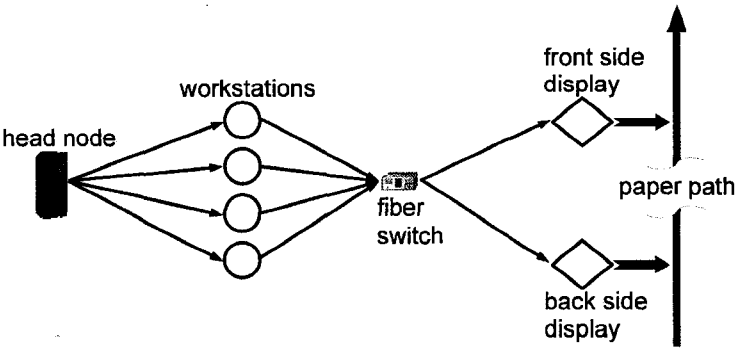


Figure 4: A conceptual model of a high performance, cluster-based imaging system.

duplex (i.e., two-sided) printer. Paper is physically moved through each press successively at high speed and cannot be immediately stopped. Consequently, if a bitmap is not readily available in the display device when it is needed, then a service interruption occurs because the printer must be stopped to accommodate the delay and the advanced paper removed from the output of the device.

The imaging system is composed of a collection of workstations dedicated to image rasterization, controlled by a separate master workstation called the head node. Individual sheetsides are transferred to the head node, where they are dispatched by the centralized image dispatcher to one of the dedicated workstations for rasterization. Each sheetside completely describes an entire display image suitable for the output device, and is expressed in a logical page description language (pdl) that must be transformed into a bitmap suitable for the display device.

Input sheetsides are queued for rasterization in the Head Node Input Queue (HNIQ). The centralized image dispatcher assigns sheetsides from the HNIQ to workstations for rasterization. After assignment, the head node places the sheetside in a queue for a transmitter that will then transmit the sheetside to its destination. The size of the transmitter queue is limited to only two sheetsides and the transmitter may only transfer one sheetside at a time. Further, once a sheetside has been placed into the transmitter queue, the destination workstation for the sheetside can no longer be modified. Each workstation has a finite capacity input buffer for storing sheetsides prior to their rasterization. Therefore, before the incoming sheetside can be placed in the transmit queue, the head node must ensure

that sufficient capacity exists in the input buffer of the receiving workstation to acquire the file. If there is sufficient input buffer capacity, then the sheetside may be queued up for transmission.

It is assumed that M heterogeneous dedicated workstations are available to convert pdl sheetsides to bitmaps. Each workstation is interconnected to the head node via a 1 Gbit ethernet network and interconnected to the two output display devices using a 4Gbit fiber channel. The memory of each workstation is divided into two blocks, where one block is used to store sheetside pdl files (the input to rasterization) and the other block is used to store output bitmaps. The sheetsides in the input block are accessed in a FIFO fashion.

When a workstation completes the rasterization of a sheetside, notification of the completion is sent to the head node, and the appropriate display device. When the display device is ready to display the bitmap, it retrieves the completed bitmap directly from the output buffer of the workstation where the rasterization was performed. Each display device has an input buffer with sufficient capacity to store two bitmaps, i.e., the bitmap currently being displayed and the next bitmap to be displayed. In this system, all bitmap files are assumed to be the same size, and the time required to display each bitmap is assumed constant.

3.3 Model of Rasterization Completion Time

The mathematical model of this system defines a method for calculating the deadline for a given sheetside, and a method for determining the estimated rasterization completion time for a sheetside on a given workstation in the system at a specific point in time. The completion of every sheetside is subject to a hard deadline. That is, to prevent a service interruption, each sheetside must complete rasterization, and be available for consumption in the input buffer of the appropriate display device by its given deadline. To calculate the deadline for a sheetside, let t_0 be the absolute wall-clock start time for both display devices. Starting at t_0, each display will require a new bitmap every $t_{display}$ seconds, where $t_{display}$ is the time required to display a bitmap on the device. Sheetsides are numbered starting with 1. Incoming sheetsides are divided between the two displays such that the odd numbered

sheetsides go to display 1 and the even numbered sheetsides go to display 2 (see Figure 4). For the k^{th} actual sheetside of the job, denoted S_k, the bitmap must be available for printing at time $t_0 + t_{display}\left(\frac{S_k-1}{2}\right)$, if k is odd, and at time $t_0 + t_{display}\left(\frac{S_k}{2}\right)$, if k is even (note: the division by 2 is because two sides are printed simultaneously). Let t_{tran}^{bitmap} be the bitmap transfer time from any workstation to either display device. Then, the deadline for completing S_k, denoted $t_d[S_k]$, is the latest wall-clock time for a workstation to produce the bitmap for S_k.

$$t_d[S_k] = \begin{cases} t_0 + t_{display}\left(\frac{S_k-1}{2}\right) - t_{tran}^{bitmap} & \text{if } S_k \text{ is odd;} \\ t_0 + t_{display}\left(\frac{S_k}{2}\right) - t_{tran}^{bitmap} & \text{if } S_k \text{ is even} \end{cases} \tag{11}$$

The value of $t_d[S_k]$ will be used later to determine the availability of output buffer space on a workstation. For this purpose, the deadline equation needs to be expressed in terms of the ordering of sheetsides on a given workstation. Let BQ_i^j be the i^{th} sheetside to have entered the input queue of workstation j for a given job. We define the following operator $\text{num}(BQ_i^j)$ that evaluates to the actual sheetside number of sheetside BQ_i^j, i.e., $S_k = \text{num}(BQ_i^j)$. Then S_k in Equation 11 can be replaced by $\text{num}(BQ_i^j)$. Note that S_k and BQ_i^j represent the same physical sheetdside and by using the num operator these notations can be used interchangeably.

The estimated rasterization *completion* time for a sheetside is composed of the earliest possible time that rasterization can begin and the estimated rasterization time. Let $t_{start}[BQ_i^j]$ be the earliest possible rasterization start time for sheetside BQ_i^j, let $ERT[BQ_i^j]$ be the estimated rasterization time for sheetside BQ_i^j, and let $t_{comp}[BQ_i^j]$ be the estimated rasterization completion time for a given sheetside BQ_i^j on workstation j. Then $t_{comp}[BQ_i^j]$ can be calculated as:

$$t_{comp}[BQ_i^j] = t_{start}[BQ_i^j] + ERT[BQ_i^j]. \tag{12}$$

The estimated rasterization time, $ERT[BQ_i^j]$, for each sheetside is assumed known based on empirical data. There are many well-known techniques for gathering these execution time estimates from empirical data [46, 54, 61, 91, 108]. The start time for rasterization depends

on several factors: when the sheetside was transferred to the workstation, when the previous sheetside assigned to the workstation completed, and the availability of the output buffer of the workstation. The rasterization for a sheetside starts only if the output buffer has enough capacity to accomodate the resultant bitmap. During the rasterization process the amount of memory required to store a bitmap remains reserved in the output buffer. The memory held by a sheetside in the input buffer is released when rasterization for that sheetside is completed.

To determine when a sheetside was (or will be) transferred to a workstation, we have to consider all other sheetsides in the HNIQ that are ahead of it. Let S_k be the k^{th} sheetside to enter the HNIQ for a given job, where S_{k-1} is the sheetside ahead of S_k in the HNIQ. To evaluate the estimated departure time for S_k to workstation j, the input buffer capacity of workstation j must be determined. Space in the input buffer is limited by two factors: the maximum number of sheetsides ($\underline{Q}$) allowed in the input buffer and the size in bytes of the input buffer.

Recall that the estimated rasterization times are known to be only estimates of the actual rasterization times. The number of sheetsides that are allowed to queue up on any given workstation is limited to a relatively small, fixed number of pending sheetsides, to attempt to mitigate the impact of delays caused by under-estimating sheetside rasterization times. If the size of the pdl file describing sheetside S_k is less than or equal to the available input buffer capacity of workstation j, then, assuming there are fewer than Q sheetsides in the input buffer of workstation j, S_k can be immediately sent to j following the transmission of S_{k-1} out of the head node. Otherwise, S_k will be delayed at the head node (blocking S_z, $z > k$) for the amount of time required for a certain number of sheetsides previously assigned to workstation j to be rasterized, thus, creating buffer capacity sufficient to accommodate the pdl file of sheetside S_k or for the number of pending sheetsides on workstation j to be less than Q.

To calculate the available input buffer capacity at workstation j, let $\underline{\mathcal{K}}$ be the sequence of sheetsides that are in the input buffer of workstation j when the head node transmitter is ready to send sheetside S_k. Note that this will include any sheetside currently being

```
if ( (size(S_k) ≤ AC_in^j)  &  (|K| < Q ))
   t_dept^j[S_k] = t_dept^x[S_{k-1}] + t_tran^sdf[S_{k-1}];
end if
else
   BQ_i^j = first element of K;
   min_size = AC_in^j;
   files = |K|;
   while ((size(S_k) > min_size) OR (files ≥ Q))
     min_size = min_size + size(BQ_i^j);
     BQ_i^j ← next element in K;
     files = files - 1;
   end while
   t_dept^j[S_k] = t_comp^j[BQ_i^j]
end else
```

Figure 5: Pseudo-code for determining the earliest estimated departure time for sheetside S_k assigned to workstation j.

rasterized by workstation j. Let the operator $\underline{\text{size}(S_i)}$ give the size in bytes of the pdl file for sheetside S_i, and let $\underline{CAP_{in}^j}$ describe the total input buffer capacity of workstation j. Then, the available capacity of the input buffer for workstation j, denoted $\underline{AC_{in}^j}$, is:

$$AC_{in}^j = CAP_{in}^j - \sum_{\forall S_i \in \mathcal{K}} \text{size}(S_i). \tag{13}$$

Define $\underline{t_{dept}^x[S_{k-1}]}$ as the departure time of S_{k-1} for HNIQ to workstation x. Let $\underline{t_{tran}^{sdf}[S_{k-1}]}$ be the time required to transfer the sheetside description file describing S_{k-1}, from HNIQ to workstation x. If $\text{size}(S_k) \leq AC_{in}^j$ and $|\mathcal{K}| < Q$, S_k can depart at time:

$$t_{dept}^j[S_k] = t_{dept}^x[S_{k-1}] + t_{tran}^{sdf}[S_{k-1}]. \tag{14}$$

Otherwise, sheetside S_k cannot be transmitted until a sufficient number of sheetsides have been processed from the input buffer of workstation j, to ensure that these two conditions hold. If after processing some sheetside $S_m \in \mathcal{K}$ these conditions hold, then $t_{dept}^j[S_k] = t_{comp}^j[S_m]$. If $k = 1$, i.e., S_k is the first sheetside to be dispatched by the centralized image dispatcher, then S_k can depart immediately. Figure 5 presents a concise pseudo-code form for determining the earliest estimated departure time for a given sheetside.

Prior to rasterization, accurately determining when rasterization can begin for some sheetside BQ_i^j, where $S_k = num(BQ_i^j)$, must also account for possible delays incurred due to the limited capacity of the output buffer on each workstation. Consider workstation j with output buffer capacity $\underline{CAP_{out}^j}$. Because bitmaps are all assumed to be the same size, i.e., require the same number of bytes to store, the number of bitmaps that could be stored in the output buffer of any workstation is constant and known in advance. Assume that $\underline{N}$ bitmaps can be placed in the output buffer of a given workstation. Define the delay to begin processing sheetside BQ_i^j, denoted $\underline{\Delta_{out}[BQ_i^j]}$, as the time that BQ_i^j must wait after arriving at the head of the input queue of workstation j until there is sufficient capacity in the output buffer of the workstation to store the output bitmap. To quantitatively determine $\Delta_{out}[BQ_i^j]$, there are three cases to consider. First, if fewer than N sheetsides have entered input queue of workstation j, then the output buffer of workstation j cannot be full, i.e., $\Delta_{out}[BQ_i^j] = 0$. In the second case, assume that more than N sheetsides have entered the input queue of workstation j, but at the time when sheetside BQ_i^j completes there will be at least one free slot in the output buffer of workstation j, i.e., at least sheetside BQ_{i-N}^j has left the output buffer, then $\Delta_{out}[BQ_i^j] = 0$. In the final case, if the output buffer of workstation j is full when sheetside BQ_{i-1}^j completes, then BQ_i^j must wait for an opening in the output buffer before its processing can begin. Therefore, sheetside BQ_i^j will be delayed until the sheetside at the head of the output buffer of the workstation completes transmission to the raster device. The three delay cases for BQ_i^j to begin processing can be described succinctly as follows.

$$\text{Case 1: if } i < N, \Delta_{out}[BQ_i^j] = 0 \tag{15}$$

$$\text{Case 2: if } t_d[BQ_{i-N}] + t_{tran}^{bitmap} \leq t_{comp}[BQ_{i-1}^j], \Delta_{out}[BQ_i^j] = 0 \tag{16}$$

$$\text{Case 3: otherwise,} \Delta_{out}[BQ_i^j] = t_d[BQ_{i-N}^j] + t_{tran}^{bitmap} - t_{comp}[BQ_{i-1}^j] \tag{17}$$

Using $\Delta_{out}[BQ_i^j]$, we can define the estimated rasterization *start* time of sheetside BQ_i^j. That is, $t_{start}[BQ_i^j]$ occurs when two conditions are satisfied: BQ_i^j is present at the head

of the input queue on workstation j, and the output buffer of workstation j has sufficient capacity to accommodate the rasterization result. If these conditions are not satisfied, then $t_{start}[BQ_i^j]$ is defined by one of two cases. First, if there is *no* opening in the output buffer of workstation j when BQ_{i-1}^j completes and BQ_i^j is available at the head of the input buffer of workstation j, then:

$$t_{start}[BQ_i^j] = t_{comp}[BQ_{i-1}^j] + \Delta_{out}[BQ_i^j]. \tag{18}$$

In the second case, if there is an opening in the output buffer of workstation j when BQ_{i-1}^j completes and BQ_i^j is *not* in the input buffer of workstation j, then the estimated start time of BQ_i^j is equal to the arrival time of BQ_i^j in the input buffer, i.e.:

$$t_{start}[BQ_i^j] = t_{dept}[BQ_i^j] + t_{tran}^{sdf}[BQ_i^j]. \tag{19}$$

That is, as soon as BQ_i^j arrives in the input buffer of workstation j, it will be rasterized without further delay. Note that if there is no opening in the output buffer on workstation j and BQ_i^j is not in the input buffer of workstation j, then one of the previous two cases will occur some time in the future. The two equations corresponding to the two cases for the earliest rasterization start time can be combined to calculate the estimated start time for BQ_i^j as follows:

$$t_{start}[BQ_i^j] = \max\{\left(t_{comp}[BQ_{i-1}^j] + \Delta_{out}[BQ_i^j]\right), \left(t_{dept}[BQ_i^j] + t_{tran}^{sdf}[BQ_i^j]\right)\}. \tag{20}$$

Note that calculation of $t_{comp}[BQ_i^j]$ is based on a recursion because it depends on $t_{start}[BQ_i^j]$ which in turn relies on $t_{comp}[BQ_{i-1}^j]$. The recursion basis is formed with BQ_1^j, whose $t_{comp}[BQ_1^j]$ is found as:

$$t_{comp}[BQ_1^j] = ERT[BQ_1^j] + t_{dept}[BQ_1^j] + t_{tran}^{sdf}[BQ_1^j]. \tag{21}$$

That is, because BQ_1^j is the first sheetside to be rasterized on workstation j, there can be

no delays incurred from processing earlier sheetsides on this workstation.

3.4 Minimum Rasterization Completion Time Heuristic

The resource allocation heuristic described in this section assumes that the system is in a steady state of operation, i.e., some sheetsides have already been rasterized and the start time t_0 for the display devices is known. In this situation, sheetsides are dispatched to the workstation that provides the minimum rasterization completion time as defined by our mathematical model of the system. That is, $t_{comp}[S_k]$ is calculated for every workstation as if S_k were assigned to it, and the workstation that gives the minimum value is selected.

Because this is a dynamic environment, updates to the minimum rasterization completion time occur when rasterization completes for a given sheetside, i.e., the actual time required for rasterization becomes known. Immediately following a sheetside completion, a control message is sent to the head node informing it of the completion. The head node then updates the recurrence equation for calculating the completion time of any subsequent sheetsides with this new information, thus, the rasterization completion time estimates become more accurate. As a result of these updates, the minimum rasterization completion time workstation for a given sheetside may change over time, i.e., the heuristic choice is time dependent.

The Minimum Rasterization Completion Time heuristic (MRCT) begins by calculating the $t^j_{comp}[S_k]$ values for the sheetside at the head of the head node input queue (S_k) for all of the workstations in the system. Workstations are then ranked in ascending order according to their $t^j_{comp}[S_k]$ values and placed together in a table in that order.

After MRCT creates the table for ranking workstations for S_k. If there is room in the input buffer of the highest ranked workstation j, i.e., $AC^j_{in} \geq \text{size}(S_k)$ and $|\mathcal{K}| < Q$, and there is a free slot in the transmit queue, then S_k is assigned to the selected workstation and placed in the transmit queue. If any of the required conditions is not satisfied, then the conditions will be satisfied some time in the future. As each workstation completes a sheetside, the table for S_k is updated and reordered.

Because the execution time estimates for rasterization are known to be estimates of

the actual execution times, the heuristic must account for cases where the estimated rasterization completion time for S_k is significantly under-estimated. For an under-estimated rasterization completion time to be significant, another workstation in the system must have a smaller or equivalent completion time for S_k compared to the actual rasterization completion time that has been under-estimated. The time at which the under-estimate for S_k becomes significant is referred to as the invalidation time for workstation j, denoted $INVT_j$.

To calculate $INVT_j$, the head node uses the rasterization completion notifications that it receives from each workstation. Because of this feedback, the head node can calculate the earliest expected feedback time ($EEFT_j$) for the completion of the sheetside currently being rasterized on workstation j, using the start time of the rasterization and the expected rasterization execution time. Using $EEFT_j$ and the estimated completion time for S_k on workstation (j) that is estimated to complete it soonest and the next best workstation (x), the invalidation time for a given workstation j $(INVT_j)$ can be calculated as follows:

$$INVT_j = EEFT_j + \left(t^x_{comp}[S_k] - t^j_{comp}[S_k]\right). \tag{22}$$

That is, if at time $INVT_j$ the current sheetside being rasterized on workstation j has not completed, it has exceeded its estimated completion time $(EEFT_j)$ by an amount $(t^x_{comp}[S_k] - t_{comp^j}[S_k])$ that now must make workstation x the best choice for S_k.

Once the ordering of workstations has been established, the $INVT_j$ values can be calculated for each of the workstations. If any of the $INVT_j$ values are in the past, then the corresponding workstations are "invalidated." The MRCT heuristic will not consider any workstations during allocation that are currently invalidated, i.e., while a workstation is marked as invalid no sheetsides will be assigned to it. When feedback regarding a sheetside completion on workstation j is received, the $EEFT_j$, $t^j_{comp}[S_i]$, and $INVT_j$ values for the associated workstation are recalculated and the invalidation status of the workstation is reset to valid.

3.5 Bitmap Lifetime

In general, the primary goal of the system is to ensure that all incoming sheetsides are rasterized and available by their deadline for display. To assess whether a sheetside is available by its deadline, we defined a new measure known as "bitmap lifetime." Bitmap lifetime is measured as the time difference between when the rasterized image is made available in some workstation's output buffer and when the raster display consumes the image from the system, i.e., the amount of time that a bitmap *lives* in an output buffer of the system before it is displayed.

When the bitmap lifetime is greater than t_{tran}^{bitmap}, the bitmap will arrive in time to be displayed without disrupting the system. If lifetime of a bitmap is not greater than t_{tran}^{bitmap}, then the bitmap will not arrive in time to be displayed by its deadline, and the system is disrupted. To represent this in our simulations, we say that whenever a bitmap lifetime is not greater than t_{tran}^{bitmap}, it is given the value t_{tran}^{bitmap} and will cause a system disruption.

The MRCT heuristic attempts to minimize the rasterization completion time of a sheetside S_k based on its most current estimates of the system state. Alternatively, we can view this minimization as an attempt to maximize the bitmap lifetime of S_k. Because the deadline for the completion of each sheetside is fixed relative to t_0, by minimizing the estimated completion time for each sheetside we are maximizing the difference between the deadline for a sheetside and its completion time.

3.6 Quantifying Robustness

3.6.1 Overall Robustness Metric

To derive a robustness metric suitable for this environment, we follow the FePIA procedure presented in [3]. In step 1 of the FePIA procedure, we describe quantitatively the requirement that makes the system robust. Intuitively, the system is robust if no service interruptions occur. That is, the performance measure of interest in this system is the completion time of each sheetside. If the completion time for each sheetside is less than its deadline, then the system can be considered to be robust.

In step 2, we identify the uncertainty in system parameters that may impact our performance feature of interest. In this system, there are two sources of uncertainty in system parameters that may cause our rasterization completion times to increase (possibly violating our robustness requirement). First, we have assumed throughout this work that our rasterization execution time estimates may differ substantially from actual rasterization execution times. Second, the arrival order of sheetsides to the system is unknown. That is, we do not know in advance when complex sheetsides will arrive for rasterization.

Step 3 of the FePIA procedure requires that we identify the impact of uncertainty in system parameters on our performance feature of interest. In this case, rasterization completion time estimates are created based on a sum of rasterization execution estimates that each may contain errors. Consequently, the uncertainty in rasterization execution times will directly impact rasterization completion time estimates.

The last step is to conduct an analysis to determine the smallest collective change in assumed values for system uncertainty parameters (from step 2) that would cause the performance feature of step 1 to violate the robustness requirement. To determine the robustness of an overall resource allocation, we will first quantify the robustness of the completion time estimate for a single sheetside BQ_i^j. Let $\underline{\mathcal{B}_i^j}$ be the set of sheetsides pending on machine j before and including BQ_i^j. The rasterization execution time estimates for any of the sheetsides in $\mathcal{B}_i^j$ may be a source of uncertainty in calculating the rasterization completion time estimate for BQ_i^j. The completion time estimates for these sheetsides are coupled because they are executed sequentially on the same workstation. That is, if the rasterization time of the first sheetside is longer than expected, then this will impact the completion time calculations for each of the subsequent sheetsides on that workstation.

From [3], we can use a geometric interpretation of our robustness requirement where the rasterization completion time estimate is a single point in an $\mathcal{N}$-dimensional space. Each of the $\mathcal{N}$ dimensions in this space corresponds to a member of $\mathcal{B}_i^j$ and we wish to find the smallest distance from our rasterization completion time estimate to the surface defined by our robustness requirement. This distance defines the smallest collective increase in assumed system parameters that would cause our robustness requirement to be violated (based on a

Euclidean distance). Because the completion time estimates for each sheetside are coupled, the true geometric interpretation of the robustness requirement must be expressed as an $\mathcal{N}$ dimensional surface. Without knowing the exact shape of this surface, we cannot calculate the shortest distance from our known point to the surface. Thus, to calculate this distance in our Robust MRCT heuristic (presented in Subsection 3.6.2, we have chosen to approximate this surface by a hyperplane. Using the equation for the distance from a point to a hyperplane [3] we can find the robustness of the completion time for sheetside BQ_i^j given a current assignment of sheetsides $\underline{\mu}$ at time t, denoted $r_\mu(BQ_i^j, t)$ as:

$$r_\mu\left(BQ_i^j, t\right) = \frac{t_d[BQ_i^j] - t_{comp}[BQ_i^j]}{\sqrt{\mathcal{B}_i^j}}. \tag{23}$$

To find the overall robustness of a resource allocation at time t, we identify the smallest robustness value for each workstation and then compare this smallest value across all workstations. We combine the $r_\mu\left(BQ_i^j, t\right)$ values for all $\mathcal{B}_i^j$ at time t to form the robustness metric for workstation j at time t, denoted $\rho_\mu^j(t)$ as follows:

$$\rho_\mu^j(t) = \min \forall i \in \mathcal{B}_i^j \left\{ r_\mu\left(BQ_i^j, t\right) \right\}. \tag{24}$$

The smallest of the $\rho_\mu^j(t)$ values over all workstations defines the *local* robustness value for the system at time t. That is,

$$\rho_\mu(t) = \min \forall j \left\{ \rho_\mu^j(t) \right\}. \tag{25}$$

Because any service interruption is unacceptable in this environment, we have chosen to use the smallest local robustness value encountered throughout a simulation run as the *overall* robustness metric. Formally, we can express the overall robustness metric value for a particular resource allocation in this system as:

$$\rho_\mu = \min \forall t \left\{ \rho_\mu(t) \right\} \tag{26}$$

3.6.2 Robust MRCT

The robustness of each sheetside completion time estimate can be used during resource allocation to aid in selecting the best workstation to process a given sheetside. In the MRCT heuristic presented earlier, we can replace the rasterization completion estimate determined for each workstation with the robustness metric.

If we compare the bitmap lifetime metric with robustness, then we can see that the numerator in the robustness equation (23) is actually an estimate of bitmap lifetime at time t. The fundamental difference between the two calculations is the denominator of the robustness equation that attempts to account for the multiple uncertainty parameters in the bitmap lifetime estimate. However, in this environment, because the number of sheetsides that can be buffered in the input queue of each workstation is limited, the magnitude of the denominator in the robustness equation is also limited.

3.7 Simulation Setup

To evaluate our heuristics, we created a simulation model of a real printing system using the OpNet simulation environment [76]. Each simulation run consisted of a rasterization job that included on the order of 100,000 sheetsides. The simulation was executed with a head node connected by a gigabit ethernet network to six workstations used to process the incoming job. The workstations are connected to the two raster display devices by a four gigabit fiber channel. It is assumed that the raster display device requires 0.11 seconds to display each output bitmap.

Each sheetside rasterization time estimate is modeled by sampling one of two normal distributions. The first distribution was chosen to have a mean of 0.01 seconds and a standard deviation of 20% of the mean. The second distribution was chosen to have a mean of 0.85 seconds and a standard deviation of 20% of the mean. In our simulations, the ratio of 0.85 second sheetsides to 0.01 second sheetsides was chosen such that the average rasterization time was 0.22 seconds. This average rasterization time was chosen to match the processing time required to output a single bitmap from each display device. Using the chosen ratio of sheetsides, for every rasterization time sampled from the 0.85 second mean

distribution there are three sheetsides selected from the 0.01 second mean distribution.

For comparison, we implemented a round-robin heuristic that was run on identical simulations to that of our MRCT heuristic and Robust MRCT heuristic. Round-robin tries to assign the same number of sheetsides to each workstation in the cluster by defining an arbitrary fixed ordering of the workstations and repeatedly assigning one sheetside to each workstation in the ordering as buffer sizes permit [101]. If there is insufficient capacity in the input buffer of any workstation j or there are greater than Q sheetsides in the input buffer already, then round-robin waits until both of these conditions are satisfied on workstation j so that the machine ordering is obeyed. Consequently, round-robin ignores the current workload on each of the workstations, instead relying on a strict "balanced" ordering of sheetside assignments to fairly assign the workload among machines.

Although the simulation study did not attempt to directly evaluate a startup strategy for starting the displays, the simulation required some startup to begin execution. For this simulation study, we chose a simplistic strategy where the sheetsides are allocated to workstations in a round-robin fashion, prior to starting the displays, until all of the workstation output buffers are full. At this point, the displays are turned on and the centralized image dispatcher begins to use one of the three studied heuristics to allocate the remaining sheetsides for the remainder of the simulation: Round Robin, MRCT, or Robust MRCT.

3.8 Simulation Results

Figure 6 presents the results of the simulation study in terms of bitmap lifetime. The x axis of each plot represents the simulation time in seconds and the y axis represents the bitmap lifetime in seconds. The plots show the bitmap lifetime values for each bitmap consumed by the system. As described in Section 3.5, if the lifetime of a bitmap is not greater than t_{tran}^{bitmap}, then the display device will have to stop to wait for the bitmap to become available—which is unacceptable in practice. In a large scale production printing environment, the paper where the raster device is displaying the images cannot be immediately stopped to wait for bitmaps to become available. Attempting to abruptly stop the paper may ruin the result,

e.g., by tearing the paper.

In each of the plots of Figure 6, at the beginning of each simulation the bitmap lifetimes are high relative to the mean bitmap lifetime. These artificially high values occur before t_0, i.e., during this time the displays have not started to consume bitmaps. Thus, the initial bitmap lifetimes are equal to the time required to fill up the output buffers on all of the workstations prior to starting the display device.

The simulation required that each heuristic rasterize the same number of sheetsides on the same set of workstations (four in this study), where the output was consumed by two displays. The round-robin heuristic is able to complete the entire run in only slightly more time than both of the MRCT heuristics, however, it did experience a significant number of service interruptions as a result of its allocation decisions. Recall that in a high-speed printing environment any interruption is considered catastrophic. In contrast, the MRCT heuristic based on bitmap lifetime is able to complete the entire run with no interruptions. For the MRCT heuristic, bitmap lifetimes were in the range of [1.64s,16s] throughout the simulation. These results demonstrate the utility of the mathematical model to estimate rasterization completion times within the context of a resource allocation heuristic.

Finally, the results of our Robust MRCT heuristic surpass that of the bitmap lifetime based MRCT heuristic, as can be seen by comparing the plots of the two heuristic results. That is, the mean bitmap lifetime for sheetsides in the Robust MRCT heuristic is higher than that of the MRCT heuristic. This implies, that the Robust MRCT heuristic is capable of tolerating more additional complex sheetsides without interrupting service than the bitmap lifetime based MRCT heuristic. Because the Robust MRCT heuristic uses the same rasterization completion time model as MRCT, the improvements of Robust MRCT can be attributed to the use of robustness in the heuristic in place of bitmap lifetime.

3.9 Related Work

According to the literature, the problem of workload distribution considered in our research falls into the category of dynamic resource allocation, assuming that multiple invocations of a resource allocation heuristic are overlapped in time with task arrivals. The general

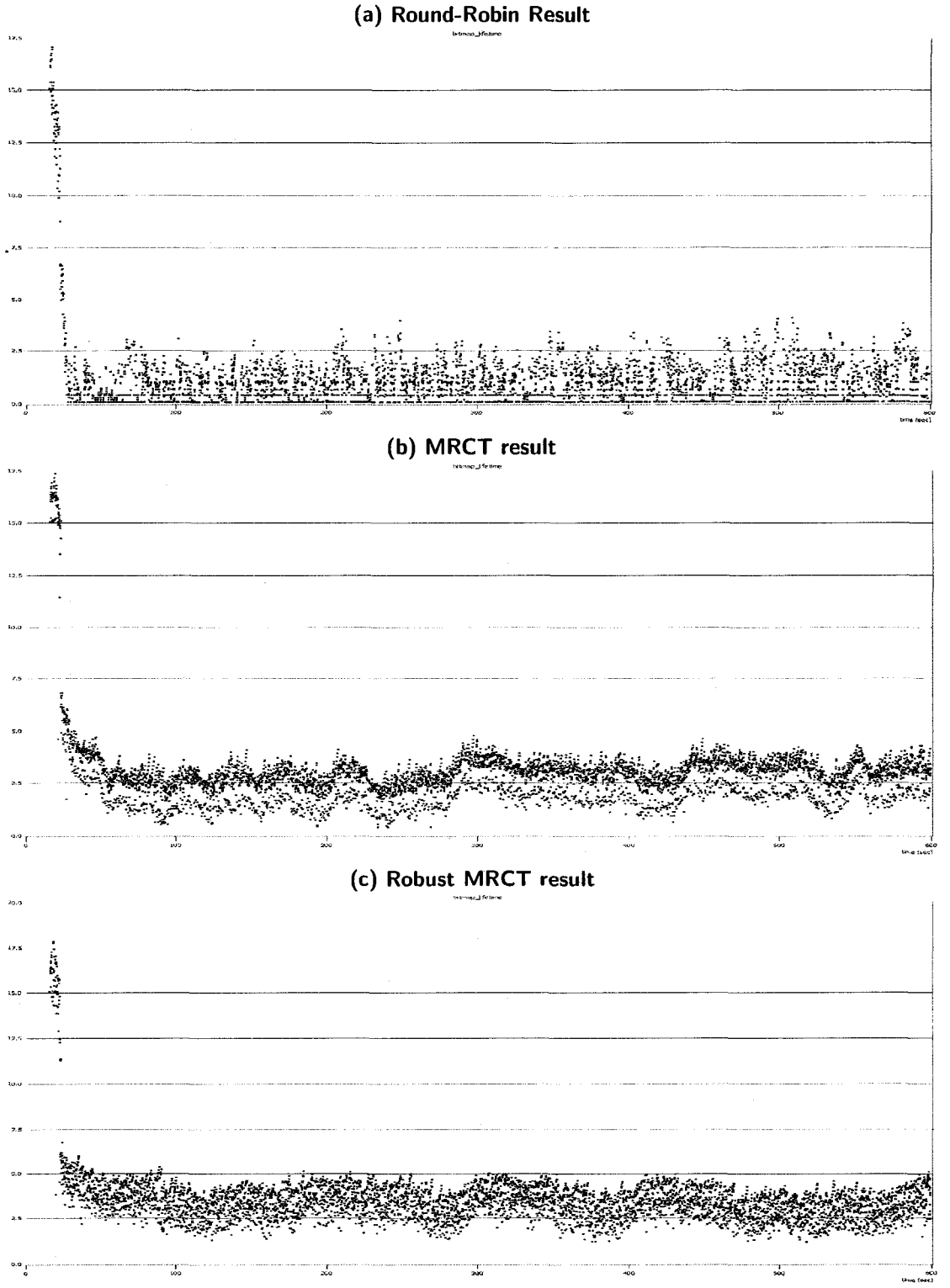


Figure 6: Sample plots of the results for the three heuristics (a) round-robin, (b) MRCT, and (c) Robust MRCT.

problem of dynamically allocating a class of independent tasks onto heterogeneous computing systems was studied in [66]. The primary objective in [66] was to minimize system makespan, i.e., the total time required to complete all tasks sent for mapping. This objective is very different from the primary objective in our work: complete rasterization of each sheetside in a given job before its assigned deadline. Our MRCT heuristic attempts to map each sheetside to its estimated minimum rasterization completion time workstation, which is analogous to the MCT heuristic of [66] attempting to map each task to its minimum completion time machine. However, the method of computing a completion time in [66] does not take into account the impacts of buffering tasks, communication links, etc. Furthermore, that study assumes no deviation of the actual time to compute a task from its estimated time to compute (ETC) value, i.e., the performance predicted by a resource allocation heuristic is assumed to match the actual performance. In our simulations, the heuristics are provided with estimated execution times that can differ from the "simulated actual" execution times. In our MRCT approach, rasterization completion time estimates for a sheetside are continuously updated with the most current information regarding the "simulated actual" sheetside completion times.

In [69], an end-to-end quality of service system is described for a distributed real-time embedded system. The authors define quality of service within an embedded system loosely as, "how well an application performs its function." The authors advocate an approach where resource allocation decisions are dynamically adapted to changes in the environment based on the coordinated monitoring and control of constrained system resources. Our work is an application of this methodology within a specific real-time imaging system. In the terminology of [69], the resource management techniques of our research are appropriate for use in the System Resource Manager role of the muli-layer resource management architecture.

In [52], a number of resource allocation heuristics for a class of independent tasks were tested on a homogeneous cluster of eight DEC Alpha workstations running Digital Unix. The set of presented heuristics includes the following five: round-robin; round-robin with clustering; minimal adaptive; continual adaptive; and first-come first-served. None of these

heuristics built a prediction model.

The robustness requirement in this work differs substantially from our earlier work on robustness in a dynamic environment [72]. In [72], the robustness requirement was expressed in terms of the overall resource allocation, i.e., expressed in terms of the entire allocation. In this work, each sheetside has an individual deadline, thus, the robustness metric must be expressed in terms of individual sheetsides. In [93], each dynamically arriving task is assigned its own deadline relative to its arrival time. However, that work assumes that stochastic information is available regarding the possible execution times of tasks. In this environment, we are only provided with a deterministic estimate of task execution times and this stochastic information is unavailable.

3.10 Conclusion

The goal of this research was to rasterize dynamically arriving sheetsides (i.e., execute tasks) before an assigned deadline for each sheetside so that service interruptions can be avoided during execution. We presented a mathematical model suitable for determining an estimate of rasterization completion times in a dynamic environment where task execution times are uncertain. We used the mathematical model to design the MRCT resource management heuristics that clearly outperformed a commonly used approach, i.e., round robin. Further, this mathematical model was used as the basis for deriving a robustness metric suitable for this environment. We presented an extension of our MRCT heuristic, Robust MRCT, that successfully utilized this robustness metric during resource allocation to surpass the results of our bitmap lifetime based approach.

CHAPTER 4

ROBUST DYNAMIC RESOURCE ALLOCATION HEURISTICS

4.1 Introduction

Heterogeneous parallel and distributed computing is defined as the coordinated use of compute resources—each with different capabilities—to optimize certain system performance features. Heterogeneous systems may operate in an environment where system performance degrades due to unpredictable circumstances or inaccuracies in estimated system parameters. The robustness of a computing system can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed [3]. Determining an assignment and scheduling of tasks to machines in a heterogeneous computing system (i.e., a mapping or resource allocation) that maximizes the robustness of a system performance feature against perturbations in system parameters is an important research problem in resource management.

This research focuses on a dynamic heterogeneous mapping environment where task arrival times are not known *a priori*. A mapping environment is considered dynamic when tasks are mapped as they arrive, i.e., in an on-line fashion [66]. The general problem of optimally mapping tasks to machines in heterogeneous parallel and distributed computing environments has been shown in general to be NP-complete (e.g., [32, 42, 50]). Thus, the development of heuristic techniques to find a near-optimal solution for the mapping problem is an active area of research (e.g., [2, 12, 13, 26, 39, 44, 62, 66, 73, 107]).

The tasks considered in this research are assumed to be taken from a frequently executed predefined set, such as may exist in a military, lab or business computing environment. The

The research presented in this chapter was jointly conducted with my colleagues Arun Jayaseelan, Ashish Mehta, and Bin Ye [70–72].

estimated time to compute (ETC) values of each task on each machine are assumed to be known based on user supplied information, experiential data, task profiling and analytical benchmarking, or other techniques (e.g., [1,45,46,56,67,109]). Determination of ETC values is a separate research problem, and the assumption of such ETC information is a common practice in mapping research (e.g., [46,54,56,61,91,108]).

For a given set of tasks, estimated makespan is defined as the completion time for the entire set of tasks based on ETC values. However, these ETC estimates may deviate from actual computation times; e.g., actual task computation times may depend on characteristics of the input data to be processed. For this research, the actual makespan of a resource allocation is required to be robust against errors in estimated task execution times. Two variations to this basic problem are considered in this work.

The first problem variation (robustness constrained) focuses on determining a dynamic mapping for a set of tasks that minimizes the estimated makespan (using the estimated ETC values) while still being able to tolerate a quantifiable amount of variation in the ETC values of the mapped tasks. Therefore, the goal of heuristics in this problem variation is to obtain a mapping that minimizes makespan while maintaining a certain level of robustness at each mapping event.

In the second problem variation (makespan constrained), the goal of the heuristics is to maximize the robustness of a resource allocation while ensuring that the makespan for the resource allocation is below a specified limit. Maximizing robustness in this context is equivalent to maximizing the amount of tolerable variation that can occur in ETC times for mapped tasks while still ensuring that a makespan constraint can be met by the resource allocation.

Dynamic mapping heuristics can be grouped into two categories: immediate mode and batch mode [66]. Immediate mode heuristics *immediately* map a task to some machine in the system for execution upon the task's arrival. In contrast, batch mode heuristics accumulate tasks until a specified condition is satisfied before mapping tasks—e.g., a certain number of tasks accumulate, or a specified amount of time elapses. When the specified condition is satisfied a mapping event occurs and the entire batch of tasks is considered for mapping.

A pseudo-batch mode can be defined where the batch of tasks considered for mapping is determined upon the arrival of a new task (i.e., a mapping event occurs) that consists of all tasks in the system that have not yet begun execution on some machine and are not next in line to begin execution, i.e., previously mapped but unexecuted tasks can be remapped.

One of the areas where this work is directly applicable is the development of resource allocations in enterprise systems that support transactional workloads sensitive to response time constraints, e.g., time sensitive business processes [75]. Often, the service provider in these types of systems is contractually bound through a service level agreement to deliver on promised performance. The dynamic robustness metric can be used to measure a resource allocation's ability to deliver on a performance agreement.

The contributions of this chapter include:

1. a model for quantifying dynamic robustness,
2. heuristics for solving the two resource management problem variations,
3. simulation results for the proposed heuristics for each problem variation, and
4. a mathematical bound on the performance feature for each of the resource management problem variation.

The remainder of the chapter is organized as follows. Section 4.2 formally states the investigated research problem. Section 4.3 describes the simulation setup. Heuristic solutions to the robustness constrained problem variation of the presented problem including an upper bound on the attainable robustness value are presented and evaluated in Section 4.4. Section 4.5 presents heuristics for the makespan constrained problem variation of the dynamic robustness problem along with their evaluation and a bound on the performance feature. Related work is considered in Section 4.6.

4.2 Problem Statement

In this study, T independent tasks (i.e., there is no inter-task communication) arrive at a mapper dynamically, where the arrival times of the individual tasks are not known in

advance. Arriving tasks are each mapped to one machine in the set of M machines that comprise the heterogeneous computing system. Each machine is assumed to execute a single task at a time (i.e., no multitasking). In this environment, the robustness of a resource allocation must be determined at every mapping event—recall that a mapping event occurs when a new task arrives to the system. Let $T(t)$ be the set of tasks either currently executing or pending execution on any machine at time t, i.e., $T(t)$ does not include tasks that have already completed execution. Let $F_j(t)$ be the predicted finishing time of machine j for a given resource allocation μ based on the given ETC values. Let $MQ_j(t)$ denote the subset of $T(t)$ previously mapped to machine j's queue and let $scet_j(t)$ denote the starting time of the currently executing task on machine j. Mathematically, given some machine j

$$F_j(t) = scet_j(t) + \sum_{\forall i \in MQ_j(t)} ETC(i,j). \tag{27}$$

Let $\beta(t)$ denote the maximum of the finishing times $F_j(t)$ for all machines at time t—i.e., the predicted makespan at time t. Mathematically,

$$\beta(t) = \max_{\forall j \in M} \{F_j(t)\}. \tag{28}$$

The robustness metric for this work has been derived using the procedure defined in [3]. In our current study, given uncertainties in the ETC values, a resource allocation is considered robust if, at a mapping event, the *actual* makespan is no more than τ seconds greater than the *predicted* makespan. Thus, given a resource allocation μ at time t, the robustness radius $r_\mu(F_j(t))$ of machine j can be quantitatively defined as the maximum collective error in the estimated task computation times that can occur where the actual makespan will be within τ time units of the predicted makespan. Mathematically, building on a result in [3],

$$r_\mu(F_j(t)) = \frac{\tau + \beta(t) - F_j(t)}{\sqrt{|MQ_j(t)|}}. \tag{29}$$

The robustness metric $\rho_\mu(t)$ for a given mapping μ is simply the minimum of the robustness

radii over all machines [3]. Mathematically,

$$\rho_\mu(t) = \min_{\forall j \in M} \{r_\mu(F_j(t))\}. \tag{30}$$

With the robustness metric defined in this way, $\rho_\mu(t)$ corresponds to the collective deviation from assumed circumstances (relevant ETC values) that the resource allocation can tolerate and still ensure that system performance will be acceptable, i.e., the actual makespan will be within τ time units of the predicted makespan.

For the robustness constrained problem variation, the dynamic robustness metric is used as a constraint. Let α be the minimum acceptable robustness of a resource allocation at any mapping event; i.e., the constraint requires that the robustness metric at each mapping event be at least α. Thus, the goal of the heuristics in the robustness constrained problem variation is to dynamically map incoming tasks to machines such that the total makespan is minimized, while maintaining a robustness of at least α i.e., $\rho_\mu(t) \geq \alpha$ for all mapping events. The larger α is, the more robust the resource allocation is.

For the makespan constrained problem variation, let T_e be the set of all mapping event times. The robustness value of the final mapping is defined as the smallest robustness metric that occurs at any mapping event time in T_e. The primary objective of heuristics in the makespan constrained problem variation is to maximize the robustness value, i.e.,

$$\text{maximize}\left(\min_{\forall t_e \in T_e} \rho_\mu(t_e)\right). \tag{31}$$

In addition to maximizing robustness, heuristics in this problem variation must complete all T incoming tasks within an overall makespan constraint (γ). Therefore, the goal of heuristics in this problem variation is to dynamically map incoming tasks to machines such that the robustness value is maximized while completing all tasks within an overall makespan constraint (based on ETC values).

4.3 Simulation Setup

The simulated environment consists of $T = 1024$ independent tasks and $M = 8$ machines for both the problem variations. This number of tasks and machines was chosen to present a significant mapping challenge for each heuristic and to make an exhaustive search for an optimal solution infeasible (however, the presented techniques can be applied to environments with different number of tasks and machines). As stated earlier, each task arrives dynamically and the arrival times are not known *a priori*. For the robustness constrained problem variation, 100 different ETC matrices were generated, 50 with high task heterogeneity and high machine heterogeneity (HIHI) and 50 with low task heterogeneity and low machine heterogeneity (LOLO) ([26]). While for the makespan constrained problem variation, 200 different ETC matrices were generated, 100 each for HIHI, and LOLO. The larger number of ETC matrices (for the makespan constrained problem variation) was needed to produce statistically reliable results. The LOLO ETC matrices model an environment where different tasks have similar execution times on a machine and also, the machines have similar capabilities, e.g., a cluster of workstations employed to support transactional data processing. In contrast, the HIHI ETC matrices model an environment where the computational requirements of tasks vary greatly and there is a set of machines with diverse capabilities, e.g., a computational grid comprising of SMPs, workstations, and supercomputers, supporting fast compilations of small programs as well as time-consuming complex simulations.

All of the ETC matrices generated were inconsistent (i.e., machine A being faster than machine B for task 1 does not imply that machine A is faster than machine B for task 2) [26]. All ETC matrices were generated using the gamma distribution method presented in [4]. The arrival time of each incoming task was generated according to a Poisson distribution with a mean task inter-arrival rate of eight seconds. In order to accentuate the difference in performance of the pseudo-batch mode heuristics in the robustness constrained problem variation, the mean task inter-arrival rate was decreased to six seconds.

In the gamma distribution method of [4], a mean task execution time and coefficient of variation (COV) are used to generate ETC matrices. In the robustness constrained

problem variation, the mean task execution time was set to 100 seconds while, for the makespan constrained problem variation, the mean task execution time was 120 seconds. For both problem variations, a COV value of 0.9 was used for HIHI and a value of 0.3 was used for LOLO. The value of τ chosen for this study was 120 seconds. The performance of each heuristic, was studied across all simulation trials, i.e., a trial corresponds to a different ETC matrix.

4.4 Robustness Constrained Heuristics

4.4.1 Heuristics Overview

Five immediate mode and five pseudo-batch mode heuristics were studied for this variation of the problem. For the task under consideration, a feasible machine is defined to be a machine that will satisfy the robustness constraint if the considered task is assigned to it. This subset of machines is referred to as the feasible set of machines.

4.4.2 Immediate Mode Heuristics

The following is a brief description of the immediate mode heuristics for this problem variation. Recall that in the immediate mode of heuristics, only the new incoming task is considered for mapping. Thus, the behavior of the heuristic is highly influenced by the order in which the tasks arrive.

4.4.2.1 Feasible Robustness Minimum Execution Time (FRMET)

FRMET is based on the MET concept in [26, 66, 110] where each incoming task is mapped to its minimum execution time machine regardless of the number of pending tasks on that machine. However, for each incoming task, FRMET first identifies the feasible set of machines. The incoming task is assigned to the machine in the feasible set of machines that provides the minimum *execution* time for the task. The procedure at each mapping event can be summarized as follows:

i. for the new incoming task find the feasible set of machines. If the set is empty, exit with error ("constraint violation")

ii. from the above set, find the minimum execution time machine

iii. assign the task to the machine

iv. update the machine available time

4.4.2.2 Feasible Robustness Minimum Completion Time (FRMCT)

FRMCT is based on the MCT concept in [26,66,110] where each incoming task is mapped to its minimum completion time machine. However, for each incoming task, FRMCT first identifies the feasible set of machines for the incoming task. From the feasible set of machines, the incoming task is assigned to its minimum *completion* time machine. The procedure at each mapping event can be summarized as follows:

i. for the new incoming task find the feasible set of machines. If the set is empty, exit with error ("constraint violation")

ii. from the above set, find the minimum completion time machine

iii. assign the task to the machine

iv. update the machine available time

4.4.2.3 Feasible Robustness K-Percent Best (FRKPB)

FRKPB is based on the KPB concept in [58, 66]. FRKPB tries to combine the aspects of both MET and MCT. FRKPB first finds the feasible set of machines for the newly arrived task. From this set, FRKPB identifies the k-percent feasible machines that have the smallest *execution* time for the task. The task is then assigned to the machine in the set with the minimum *completion* time for the task. For a given α the value of k was varied between 0 and 100, in steps of 12.5, for sample training data to determine the value that provided the minimum makespan. A value of $k = 50$ was found to give the best results. The procedure at each mapping event can be summarized as follows:

i. for the new incoming task find the feasible set of machines. If the set is empty, exit with error ("constraint violation")

ii. from the above set, find the top $m = 4$ machines based on execution time

iii. from the above find the minimum completion time machine

iv. assign the task to the machine

v. update the machine available time

4.4.2.4 Feasible Robustness Switching (FRSW)

FRSW is based on the SW concept in [58,66]. As applied in this research, FRSW combines aspects of both the FRMET and the FRMCT heuristics. A *load balance ratio* (LBR) is defined to be the ratio of the minimum number of tasks enqueued on any machine to the maximum number of tasks enqueued on any machine. FRSW then switches between FRMET and FRMCT based on the value of the load balance ratio. The heuristic starts by mapping tasks using FRMCT. When the ratio rises above a high set point, denoted T_{high} FRSW switches to the FRMET heuristic. When the ratio falls below a low set point, denoted T_{low} FRSW switches to the FRMCT heuristic. The values for the switching set points were determined experimentally using sample training data. The procedure at each mapping event can be summarized as follows:

i. for the new incoming task find the feasible set of machines. If the set is empty, exit with error ("constraint violation")

ii. calculate the load balance ratio (LBR)

iii. initial mapping heuristic - FRMCT

if LBR $> T_{high}$ map using FRMET

else if LBR $< T_{low}$ map using FRMCT

else if $T_{low} \leq$ LBR $\leq T_{high}$ map using previous mapping heuristic

4.4.2.5 Maximum Robustness (MaxRobust)

MaxRobust has been implemented for comparison only, trying to greedily maximize robustness without considering makespan. MaxRobust calculates the robustness radius of each

machine for the newly arrived task, assigning the task to the machine with the maximum robustness radius. The procedure at each mapping event can be summarized as follows:

i. for the new incoming task find the robustness radius for each machine, considering the previous assignments

ii. assign task to maximum robustness radius machine

iii. update the machine available time

4.4.3 Pseudo-Batch Heuristics

The pseudo-batch mode heuristics implement two sub-heuristics, one to map the task as it arrives, and a second to remap pending tasks. For the pseudo-batch mode heuristics, the initial mapping is performed by the previously described FRMCT heuristic (except for the MRMR heuristic). The remapping heuristics each operate on a set of mappable tasks; a mappable task is defined as any task pending execution that is not next in line to begin execution. The following is a brief description of the pseudo-batch mode re-mapping heuristics.

4.4.3.1 Feasible Robustness Minimum Completion Time-Minimum Completion Time (FMCTMCT)

FMCTMCT uses a variant of Min-Min heuristic defined in [50]. For each mappable task, FMCTMCT finds the feasible set of machines, then from this set determines the machine that provides the minimum completion time for the task. From these task/machine pairs, the pair that gives the overall minimum completion time is selected and that task is mapped onto that machine. This procedure is repeated until all of the mappable tasks have been remapped. The procedure at each mapping event can be summarized as follows:

i. map the new incoming task using FRMCT

ii. if set of mappable tasks is not empty

   (a) for each task, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation")

(b) for each task find the feasible machine that minimizes computation time (first Min), ignoring other mappable tasks

(c) from the above task/machine pairs, find the pair that gives the minimum completion time (second Min)

(d) assign the task to the machine and remove it from the set of mappable tasks

(e) update the machine available time

(f) repeat a-e until all tasks are remapped

4.4.3.2 Feasible Robustness Maximum Robustness-Minimum Completion Time (FMRMCT)

FMRMCT builds on concept of the Max-Min heuristic [50]. For each mappable task, FMRMCT first identifies the feasible set of machines, then from this set determines the machine that provides the minimum completion time. From these task/machine pairs, the pair that provides the maximum robustness radius is selected and the task is assigned to that machine. This procedure is repeated until all of the mappable tasks have been remapped. The procedure at each mapping event can be summarized as follows:

i. map the new incoming task using FRMCT

ii. if set of mappable tasks is not empty

(a) for each task, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation")

(b) for each task find the feasible machine that minimizes computation time (Min), ignoring other mappable tasks

(c) from the above task/machine pairs, find the pair that gives the maximum robustness radius (Max)

(d) assign the task to the machine and remove it from the set of mappable tasks

(e) update the machine available time

(f) repeat a-e until all tasks are remapped

4.4.3.3 Feasible Minimum Completion Time-Maximum Robustness (FMCTMR)

For each mappable task, FMCTMR first identifies the feasible set of machines, then from this set determines the machine with the maximum robustness radius. From these task/machine pairs, the pair that provides the minimum completion time is selected and the task is mapped to that machine. This procedure is repeated until all of the mappable tasks have been remapped. The procedure at each mapping event can be summarized as follows:

i. map the new incoming task using FRMCT

ii. if set of mappable tasks is not empty

   (a) for each task, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation")

   (b) for each mappable task find the feasible machine that gives maximum robustness radius (Max), ignoring other mappable tasks

   (c) from the above task/machine pairs, find the pair that gives the minimum completion time (Min)

   (d) assign the task to the machine and remove it from the set of mappable tasks

   (e) update the machine available time

   (f) repeat a-e until all tasks are remapped

4.4.3.4 Maximum Weighted Sum-Maximum Weighted Sum (MWMW)

MWMW builds on a concept in [90]. It combines the Lagrangian heuristic technique [28,65] for deriving an objective function with the concept of Min-Min heuristic [50] here to simultaneously minimize makespan and maximize robustness. For each mappable task, the feasible set of machines is identified and the machine in this set that gives the maximum value of the objective function (defined below) is determined. From this collection of task/machine pairs, the pair that provides the maximum value of the objective function is selected and the corresponding assignment is made. This procedure is repeated until all of the mappable tasks have been remapped.

When considering assigning a task i to machine j, let $F'_j(t) = F_j(t) + \sum ETC(i,j)$ for all tasks currently in the machine queue and the task currently under consideration. Let $\beta'(t)$ be maximum of the finishing times $F'_j(t)$ at time t for all machines. Let $r'_\mu(F'_j(t))$ be the robustness radius for machine j. Let *maxrob(t)* be the maximum of the robustness radii at time t. Given η , an experimentally determined constant using training data, the objective function for MWMW is defined as

$$s(j,t) = \eta\left(1 - \frac{F'_j(t)}{\beta'(t)}\right) + (1-\eta)\left(\frac{r'_\mu(F'_j(t))}{maxrob(t)}\right) \tag{32}$$

The procedure at each mapping event can be summarized as follows:

i. map the new incoming task using FRMCT

ii. if set of mappable tasks is not empty

   (a) for each task, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation")

   (b) for each task find the feasible machine that gives maximum value of the objective function ($s(j,t)$), ignoring other mappable tasks

   (c) from the above task/machine pairs, find the pair that gives the maximum value of $s(j,t)$

   (d) assign the task to the machine and remove it from the set of mappable tasks

   (e) update the machine available time

   (f) repeat a-e until all tasks are remapped

4.4.3.5 Maximum Robustness-Maximum Robustness (MRMR)

MRMR is provided here for comparison only as it optimizes robustness without considering makespan. When a task arrives it is initially mapped using the MaxRobust heuristic. Task remapping is performed by a variant of the Max-Max heuristic [50]. For each mappable task, the machine that provides the maximum robustness radius is determined. From these task/machine pairs, the pair that provides the maximum overall robustness radius is

selected and the task is mapped to that machine. This procedure is then repeated until all of the mappable tasks have been remapped. The procedure at each mapping event can be summarized as follows:

i. map the new incoming task using MaxRobust

ii. if set of mappable tasks is not empty

   (a) for each task find the machine that gives maximum robustness radius (first Max), ignoring other mappable tasks

   (b) from the above task/machine pairs, find the pair that gives the maximum value (second Max)

   (c) assign the task to the machine and remove it from the set of mappable task

   (d) update the machine available time

   (e) repeat a-d until all tasks are remapped

4.4.4 Lower Bound

A lower bound on makespan for the described system can be found by identifying the task whose arrival time plus minimum execution time on any machine is the greatest. More formally, given the entire set of tasks S where each task i has an arrival time of $arv(i)$, the lower bound is given by

$$LB_1 = \max_{\forall i \in S} \left((arv(i) + \min_{\forall j \in M} ETC(i,j) \right). \tag{33}$$

Unfortunately, this bound neglects any time that the task spends waiting to execute. This can be significant in highly loaded systems. Therefore, a second lower bound that considers the total computational load was also used. This bound is given by,

$$LB_2 = \frac{\sum_{i=0}^{T} \{ \min_{\forall j \in M} ETC(i,j) \}}{M}. \tag{34}$$

The lower bound on makespan can then be given by the maximum of the two bounds, i.e.,

$$LB = \max(LB_1, LB_2). \tag{35}$$

Clearly, this lower bound may not be achievable even by an optimal mapping, however, it is a tight lower bound because the case described by LB_1 is possible if a system is very lightly loaded.

4.4.5 Results

In Figures 7 through 10, the average makespan results (with 95% confidence interval bars) are plotted, along with a lower bound on makespan. Figures 7 and 8 present the makespan results for the immediate mode heuristics for HIHI and LOLO heterogeneity, respectively. While, Figures 9 and 10 present the makespan results for the pseudo-batch mode heuristics for HIHI and LOLO heterogeneity, respectively. Each of the heuristics was simulated using multiple values for the robustness constraint α. For each α the performance of the heuristics was observed for 50 HIHI and 50 LOLO heterogeneity trials. In Figures 7 through 10, the number of failed trials (out of 50) is indicated above the makespan results for each heuristic, i.e., the number of trials for which the heuristic was unable to successfully find a mapping for every task given the robustness constraint α.

The average execution times for each heuristic over all mapping events (on a typical unloaded 3GHz Intel Pentium 4 desktop machine) in all 100 trials are shown in Table I and Table II for immediate and pseudo-batch mode, respectively. For the immediate mode heuristics, this is the average time for a heuristic to map an incoming task. For the pseudo-batch mode heuristics, this is the average time for a heuristic to map an entire batch of tasks.

For the immediate mode heuristics, FRMET resulted in the lowest makespan for HIHI, and FRMET and FRSW performed the best for LOLO. The immediate mode FRMET heuristic for both HIHI and LOLO heterogeneity performed better than anticipated based on prior studies including a minimum execution time (MET) heuristic in other environment (that do no involve robustness and had different arrival rates and ETC matrices). It should

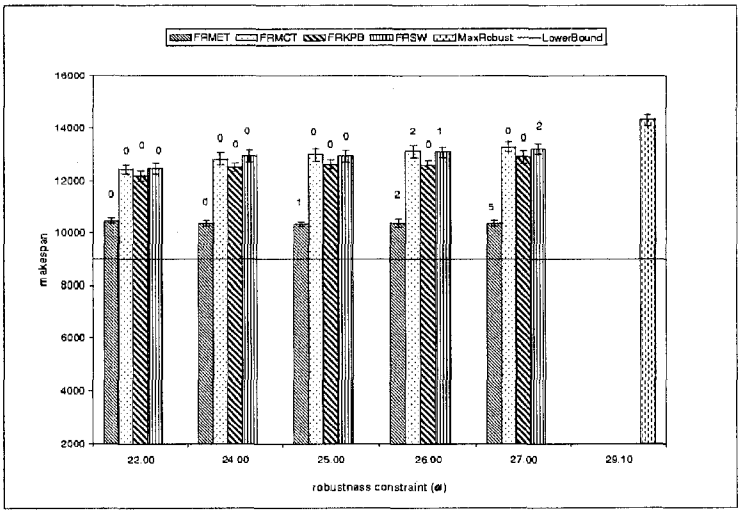


Figure 7: Simulation results of makespan for different values of robustness constraint (α) for immediate mode heuristics for HIHI heterogeneity.

be noted, however, that its performance in the HIHI case did result in multiple instances where it failed to find a mapping.

It has been shown, in general, that the minimum execution time heuristic is not a good choice for minimizing makespan for both the static and dynamic environments [26, 66], because it ignores machine loads and machine available times when making a mapping decision. The establishment of a feasible set of machines by the FRMET heuristic indirectly balances the incoming task load across all of the machines. Also, because of the highly inconsistent nature of the data sets coupled with the high mean execution time (100 seconds), FRMET is able to maintain a lower makespan compared to FRMCT.

To illustrate this, consider the ETC matrix of Table 4.4.5.

	t_0	t_1	t_2	t_3	t_4
M_1	10	150	180	150	100
M_2	100	70	170	100	150
M_3	180	100	60	140	300

Table 1: Example ETC matrix.

If the tasks arrive in the above order and the robustness constraint is $\alpha = 22$, the mapping obtained by FRMET would correspond to that of table 4.4.5,

whereas using FRMCT results in the mapping of Table 4.4.5.

M_1	$t_0(10)$	$t_4(100)$
M_2	$t_1(70)$	$t_3(100)$
M_3	$t_2(60)$	

Table 2: Example FRMET mapping result.

M_1	$t_0(10)$	$t_3(150)$
M_2	$t_1(70)$	$t_4(150)$
M_3	$t_2(60)$	

Table 3: Example FRMCT mapping result.

Thus, the makespan obtained using FRMET is 170 while that obtained using FRMCT is 220.

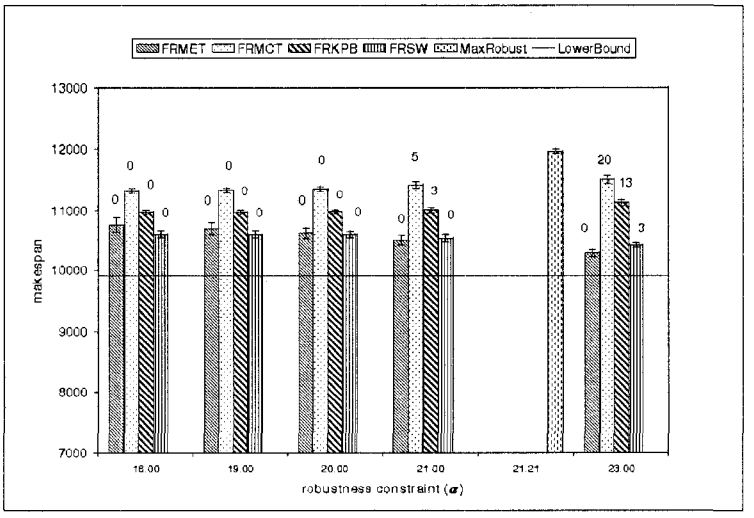


Figure 8: Simulation results of makespan for different values of robustness constraint (α) for immediate mode heuristics for LOLO heterogeneity.

Table III shows the maximum and average number of mapping events (out of a possible 1024) over successful trials (out of 50) for which the MET machine was not feasible. That is, the table values were calculated based on only the subset of the 50 trials for which FRMET could determine a mapping that met the constraint. For each of these trials, there were 1024 mapping events. Thus, even though the vast majority of tasks are mapped to their MET machine, it is important to prevent those rare cases where doing so would make the mapping infeasible.

The FRKPB heuristic performed better than FRMCT (in terms of makespan) for LOLO

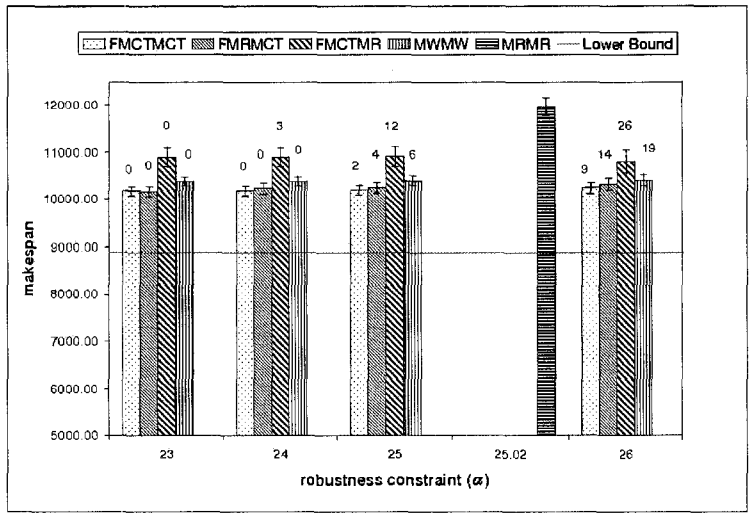


Figure 9: Simulation results of makespan for different values of robustness constraint (α) for pseudo-batch mode heuristics for HIHI heterogeneity.

heterogeneity and comparable to FRMCT for HIHI heterogeneity. FRKPB selects the k-percent feasible machines that have the smallest *execution* time for the task and then assigns the task to the machine in the set with the minimum *completion* time for the task. Thus, rather then trying to map the task to its best completion time machine, it tries to avoid putting the current task onto the machine which might be more suitable for some task that is yet to arrive. This foresight about task heterogeneity is missing in FRMCT, which might assign the task to a poorly matched machine for an immediate marginal improvement in completion time. This might possibly deprive some subsequently arriving better matched tasks of that machine, and eventually leading to a larger makespan than FRKPB.

The FRSW heuristic switches between FRMCT and FRMET depending on the LBR. In the HIHI case T_{low} was set to 0.6 and T_{high} was set to 0.9. With these values of the threshold, FRSW used FRMCT, on average, for 96% of the mapping events (out of total 1024) to map the incoming task. In the LOLO case T_{low} was set to 0.3 and T_{high} was set to 0.6. For these values of the thresholds FRSW used FRMET, on average, for 80% of the mapping events (out of total 1024) to map the incoming task. As stated earlier FRMET performs much better than FRMCT for both the HIHI and LOLO cases. Thus the better performance of FRSW, for LOLO heterogeneity, can be attributed to the fact

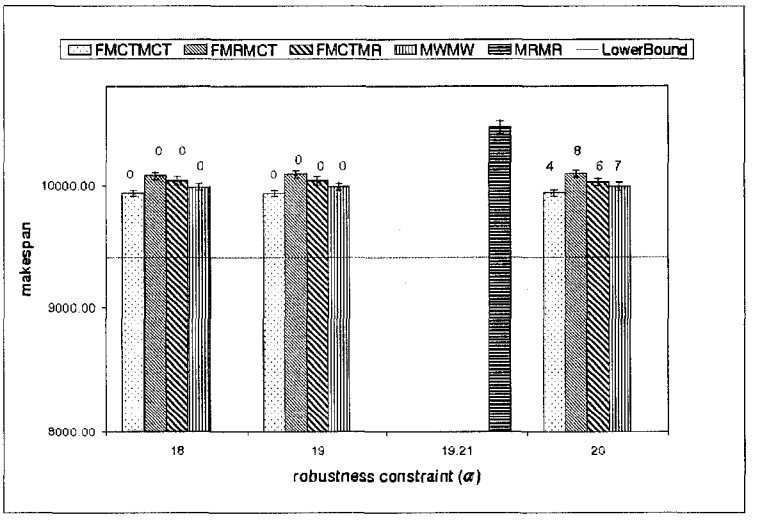


Figure 10: Simulation results of makespan for different values of robustness constraint (α) for pseudo-batch mode heuristics for LOLO heterogeneity.

that it maps a large number of tasks using FRMET as opposed to FRMCT. In contrast, for HIHI heterogeneity, a larger number of tasks are mapped using FRMCT and so the makespan is comparable to that of FRMCT.

An interesting observation was that the FRMCT heuristic was able to mantain a robustness constraint of $\alpha = 27$ for all 50 trials used in this study, but only for 48 trials when $\alpha = 26$ (for HIHI heterogeneity). This could be attributed to the volatile nature of the greedy heuristics. The looser robustness constraint ($\alpha = 26$) allowed for a paring of task to machine that was disallowed for a tighter robustness constraint ($\alpha = 27$). That is, the early greedy selection proved to be a poor decision because it ultimately led to a mapping failure.

For the HIHI case all the heuristics (except MaxRobust) failed for at least 4% (20% on average) of the trials (out of 50) for the robustness constraint achieved by MaxRobust heuristic.

When considering the performance of the pseudo-batch mode heuristics (figures 9 and 10) recall that they were evaluated across a different set of ETC matrices (mean task inter-arrival rate of six seconds as opposed to eight seconds for ETC matrices for immediate mode). The MWMW heuristic used a value of $\eta = 0.6$ for HIHI and $\eta = 0.3$ for LOLO.

For the HIHI heterogeneity trials, FMCTMCT and FMRMCT performed comparably, in terms of makespan, though FMRMCT had a higher failure rate than FRMCTMCT for high values of α. The inclusion of the concept of feasible machines helped FMCTMCT and FMRMCT maintain a high level of robustness. The FMCTMR heuristic had a higher makespan as compared to FMRMCT. The reason being the first stage choice of machines for these two-stage greedy heuristics. The FMRMCT heuristic tries to minimize the completion time in the first stage and then selects the task/machine pair that maximizes the robustness radius, as opposed to maximizing the robustness radius in stage one and then selecting the task/machine pair that minimizes the completion time as used by FMCTMR.

For the LOLO heterogeneity trials, FMCTMCT performed the best on average, while MWMW performed comparably (in terms of makespan). The motivation behind using MRMR was to greedily maximize robustness at every mapping event. As can be seen from figures 9 and 10, the MRMR heuristic was able to maintain a high level of robustness, however, it had the worst makespan among the heuristics studied.

heuristics	avg. exec. time (sec.)
FRMET	0.001
FRMCT	0.0019
FRKPB	0.0019
FRSW	0.0015
MaxRobust	0.0059

Table 4: Average execution times, in seconds, of a mapping event for the proposed immediate mode heuristics.

heuristics	avg. exec. time (sec.)
FMCTMCT	0.023
FMRMCT	0.028
FMCTMR	0.028
MWMW	0.0211
MRMR	0.0563

Table 5: Average execution times, in seconds, of a mapping event for the proposed pseudo-batch mode heuristics.

HIHI					
Robustness constraint(α)	22.00	24.00	25.00	26.00	27.00
max	41	54	73	79	88
avg	14	22	30	36	42

LOLO						
Robustness constraint(α)	18.00	19.00	20.00	21.00	21.21	22.00
max	5	10	14	26	26	56
avg	0	1	3	6	6	7

Table 6: Maximum and average number of mapping events (over successful trials) for which the MET machine was not feasible for HIHI and LOLO heterogeneity.

4.5 Makespan Constrained Heuristics

4.5.1 Heuristics Overview

Five pseudo-batch mode heuristics were studied for this research. All of the heuristics used a common procedure to identify a set of feasible machines, where a machine is considered feasible if it can execute the task without violating the makespan constraint that is, for a task under consideration, a machine is considered feasible if that machine can satisfy the makespan constraint when the task is assigned to it. The subset of machines that are feasible for the task is referred to as the feasible set of machines.

4.5.2 Heuristic Descriptions

4.5.2.1 Minimum Completion Time-Minimum Completion Time (MinCT-MinCT)

The MinCT-MinCT heuristic is similar to the FMCTMCT heuristic studied in the robustness constrained problem variation but with the new definition of the feasible machine.

4.5.2.2 Maximum Robustness-Maximum Robustness (MaxR-MaxR)

As was seen in the robustness constrained problem variation, the MRMR heuristic was able to maintain a high level of robustness, but had a higher makespan. The goal in this problem variation is to maximize the robustness at each mapping event, and hence a variation of MRMR heuristic is employed. However, unlike the MRMR heuristic, for each mappable task, MaxR-MaxR identifies the set of feasible machines. From each task's set of feasible machines, the machine that maximizes the robustness metric for the task is selected. If for

any task there are no feasible machines then the heuristic will fail. From these task/machine pairs, the pair that maximizes the robustness metric is selected and that task is mapped onto its chosen machine. This procedure is repeated until all of the mappable tasks have been mapped. The procedure at each mapping event can be summarized as follows:

i. A task list is generated that includes all mappable tasks.

ii. For each task in the task list, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation").

iii. For each mappable task (ignoring other mappable tasks), find the feasible machine that maximizes the robustness radius.

iv. From the above task/machine pairs select the pair that maximizes the robustness radius.

v. Remove the task from the task list and map it onto the chosen machine.

vi. Update the machine available time.

vii. Repeat ii-vi until task list is empty.

4.5.2.3 Maximum Robustness-Minimum Completion Time (MaxR-MinCT)

MaxR-MinCT is similar to the FMRMCT heuristic studied in robustness constrained problem variation, but with new definition of the feasible machine.

4.5.2.4 Minimum Completion Time-Maximum Robustness (MinCT-MaxR)

The MinCT-MinCT heuristic is similar to the FMCTMR heuristic studied in the robustness constrained problem variation but with the new definition of the feasible machine.

4.5.2.5 MaxMaxMinMin (MxMxMnMn)

This heuristic makes use of two sub-heuristics to obtain a mapping. It uses a combination of Min-Min with a robustness constraint (to minimize makespan while maintaining the current robustness value) and Max-Max (based on robustness) to maximize robustness while still finishing all T tasks within the overall makespan constraint. The mapping procedure begins

execution using the Min-Min heuristic with τ as the robustness level to be maintained—τ was chosen based on the upper bound discussion presented in Subsection 4.5.4. The procedure at each mapping event can be summarized as follows:

i. A task list is generated that includes all mappable tasks.

ii. Min-Min component

   (a) For each task in the task list, find the set of machines that satisfy the robustness level if the considered task is assigned to it. If the set is empty for any task, go to step iii.

   (b) From the above set of machines, for each mappable task (ignoring other mappable tasks), find the feasible machine that minimizes the completion time.

   (c) From the above task/machine pairs select the pair that minimizes completion time.

   (d) Remove the task from the task list and map it onto its chosen machine.

   (e) Update the machine available time.

   (f) Repeat a-e until task list is empty, exit.

iii. Max-Max component

   (g) A task list is generated that includes all mappable tasks (any task mapped by Min-Min in this mapping event are remapped).

   (h) For each task in the task list, find the set of feasible machines. If the set is empty for any task, exit with error ("constraint violation")

   (i) For each mappable task (ignoring other mappable tasks), find the feasible machine that maximizes the robustness metric.

   (j) From the above task/machine pairs select the pair that maximizes the robustness metric.

   (k) Remove the task from the task list and map it onto the chosen machine.

   (l) Update the machine available time.

(m) Repeat h-l until task list is empty.

iv. Update the robustness level to the new robustness value (the smallest robustness metric that has occurred).

4.5.3 Fine Tuning (FT)

A post-processing step, referred to as fine tuning (FT) was employed to improve the robustness value produced by a mapping. Fine tuning reorders tasks in the machine queues in ascending order of execution time on that machine (as done for a different problem environment in [110]), i.e., smaller tasks are placed in the front of the queues. This procedure is performed at each mapping event after executing one of the above heuristics. This procedure will not directly impact the overall finishing times of the machines, but does help in getting the smaller tasks out of the machine queues faster and thus helps reduce the numerator in equation 29, which correspondingly improves the robustness metric.

4.5.4 Upper Bound

Let the provided constant τ be the upper bound on robustness. To prove that robustness can be no higher than τ is to show that at least one machine will have at least one task assigned to it during the course of the simulation. When the first task is assigned to some machine in the system the robustness radius of that machine becomes τ. In equation 29, $\beta(t) - F_j(t)$ goes to zero for the makespan machine. Because the machine with the first and only task assigned to it is now the makespan defining machine, its robustness radius is now τ. The robustness radius of this machine defines the robustness metric for the system because it is the smallest of the robustness radii at this mapping event. Because the robustness value is defined as the smallest robustness metric over all mapping events, that value can be no greater than τ.

4.5.5 Results

In Figures 11 and 12, the average robustness value (over all mapping events) for each heuristic is plotted with their 95% confidence intervals. The average execution time of each heuristic over all mapping events in all 200 trials is shown in Table IV. Recall that the

heuristics operate in a pseudo-batch mode, therefore, the times in Table IV are the average time for each heuristic to map an entire batch of tasks.

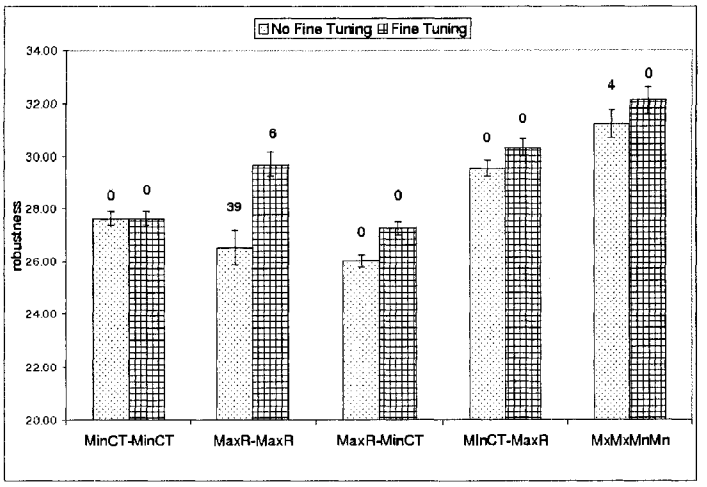


Figure 11: Average robustness value (over all mapping events) for the HIHI case with $\gamma = 14000$.

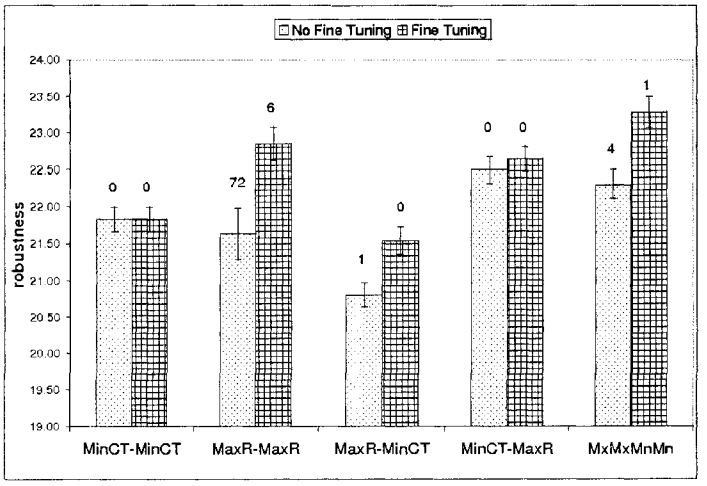


Figure 12: Average robustness value (over all mapping events) for the LOLO case with $\gamma = 12500$.

As can be seen from Figures 11 and 12, MxMxMnMn with fine tuning gives the best robustness result for both the HIHI and LOLO cases (although there is one failure). The good performance of MxMxMnMn can be attributed to the fact that the maintainable robustness value is by definition monotonically decreasing, and its approach tries to minimize makespan (using Min-Min) while maintaining the current robustness value. If that is

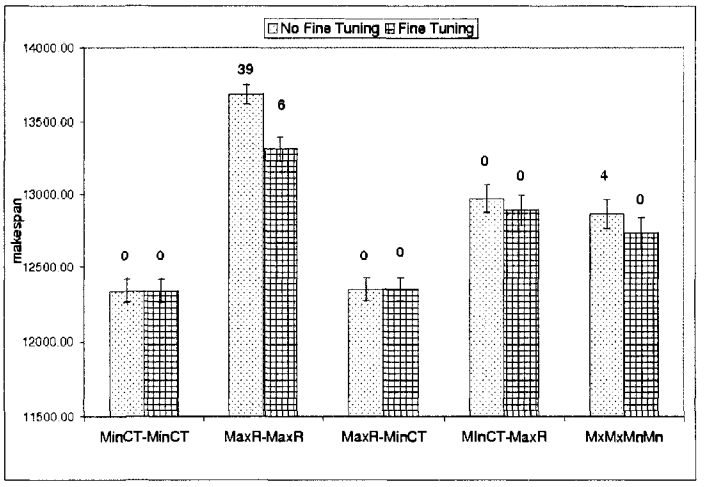


Figure 13: Average makespan for the HIHI case with $\gamma = 14000$.

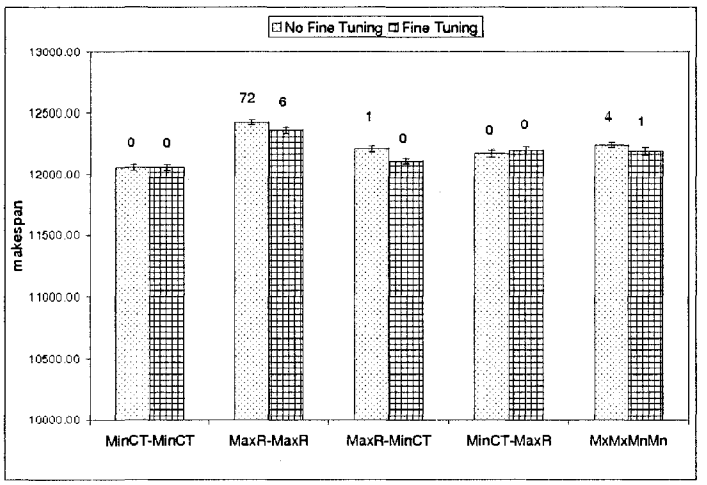


Figure 14: Average makespan for the LOLO case with $\gamma = 12500$.

not possible it instead maximizes robustness using Max-Max—attempting to minimize the degradation in the robustness value.

Although, MinCT-MinCT is able to achieve one of the best makespan (Figures 13 and 14) for both the HIHI and LOLO cases, its robustness value is not one of the best, which confirms the fact that just minimizing the finishing times of the machines does not guarantee a higher value of robustness.

The high number of failed trials for MaxR-MaxR for both the HIHI and LOLO cases can be attributed to the fact that the heuristic tries to maximize the robustness metric at all mapping events, but in doing so neglects the corresponding increase in machine finishing

times. For example, consider the following two machine system with a current robustness value of 60 and machine queues with the task execution times as shown in Table 7.

m_1:	$t_1(10)$	$t_3(10)$
m_2:	$t_2(50)$	

Table 7: Task assignments per machine queue for a given example allocation.

Assume that a new task t_4 arrives with execution times of 10 and 50 time units on machines m_1 and m_2, respectively. The MaxR-MaxR heuristic will map task t_4 to machine m_2, which increases makespan because assigning t_4 to machine m_1 would decrease the robustness metric. However, mapping t_4 to m_1 would give a new robustness metric of 80.8 that is still greater than the current robustness value of 60.

For both the HIHI and LOLO cases, MinCT-MaxR performed relatively better than MaxR-MinCT in terms of robustness. This can be explained in terms of the first stage choice of machines for this pair of two-stage greedy heuristics. MinCT-MaxR places more emphasis on directly optimizing the primary objective of maximizing the robustness value as opposed to minimizing makespan. By minimizing completion time in the second stage, MinCT-MaxR is able to stay within the overall makespan constraint while still maximizing robustness. This is evident from zero failures that occurred for MinCT-MaxR in both the LOLO and HIHI cases.

The process of fine tuning did improve the results of the heuristics, though not substantially (less than 12% for the best HIHI case and less than 5% for the best LOLO case). Further, it is possible that fine tuning when used with MxMxMnMn can cause some trials to fail to meet the makespan constraint. This occurs because fine tuning attempts to reduce the number of tasks in the machine queues by moving small tasks up in the queues. Thus, it is possible for the heuristic to maintain a higher robustness value over its execution, but at certain mapping events when the Min-Min component of the heuristic tries to map a task using a higher robustness constraint, it is likely that it will not choose the minimum completion time machine for the task because it is not feasible, which results in a higher finishing time. For example, consider a two machine system with machine queues as shown in Table 8. Assume that a new task t_4 arrives with execution times of 80 and 20 on machines m_1

m_1:	$t_1(150)$	
m_2:	$t_2(30)$	$t_3(80)$

Table 8: Example resource allocation in a two machine system.

and m_2, respectively. If MxMxMnMn maps this task using the Min-Min component with a robustness level of $\tau/\sqrt{2}$, the mapping would be: But if MxMxMnMn uses the Min-Min

m_1:	$t_1(150)$	$t_4(80)$
m_2:	$t_2(30)$	$t_3(80)$

Table 9: Example MxMxMnMn allocation result using the Max-Max portion of the heuristic.

component with a robustness level of $\tau/2$, the mapping would be:

m_1:	$t_1(150)$		
m_2:	$t_2(30)$	$t_3(80)$	$t_4(20)$

Table 10: Example MxMxMnMn allocation result using the Min-Min portion of the heuristic.

Finally, because MxMxMnMn uses a Max-Max heuristic to maximize robustness it is prone to the same issues discussed previously for the MaxR-MaxR heuristic.

4.6 Related Work

The research presented in this paper was designed using the four step FePIA procedure described in [3]. A number of papers in the literature have studied robustness in distributed systems (e.g., [17, 34, 80, 86]).

The research in [17] considers rescheduling of operations with release dates using multiple resources when disruptions prevent the use of a preplanned schedule. The overall strategy is to follow a preplanned schedule until a disruption occurs. After a disruption, part of the schedule is reconstructed to match up with the pre-planned schedule at some future time. Our work considers a slightly different environment where task arrivals are not known in advance. Consequently, in our work it was not possible to generate a preplanned schedule.

The research in [34] considers a single machine scheduling environment where processing times of individual jobs are uncertain. Given the probabilistic information about processing times for each job, the authors in [34] determine a normal distribution that approximates

heuristic	average execution time (sec.)
MinCT-MinCT	0.023
MaxR-MinCT	0.028
MinCT-MaxR	0.028
MaxR-MaxR	0.0563
MxMxMnMn	0.0457

Table 11: Average execution times, in seconds, of a mapping event for the proposed heuristics.

the flow time associated with a given schedule. The risk value for a schedule is calculated by using the approximate distribution of flow time (i.e., the sum of the completion times of all jobs). The robustness of a schedule is then given by one minus the risk of achieving sub-standard flow time performance. In our work, no such stochastic specification of the uncertainties is assumed.

The study in [80] defines a robust schedule in terms of identifying a Partial Order Schedule (POS). A POS is defined as a set of solutions for the scheduling problem that can be compactly represented within a temporal graph. However, the study considers the Resource Constrained Project Scheduling Problem with minimum and maximum time lags, (RCPSP/max), as a reference, which is a different problem domain from the environment considered here.

In [86], the robustness is derived using the same FePIA procedure used here. However the environment considered is static (off-line), as opposed to the dynamic (on-line) environment in this research. The robustness metric and heuristics employed in a dynamic environment are substantially different from those employed in [86].

4.7 Conclusion

This research presented a model for quantifying robustness in a dynamic environment. It also involved the characterization and modeling of two dynamic heterogeneous computing problem environments, and examined and compared various heuristic techniques for each of the two problem variations. This work also presented the bounds on the highest attainable value of the system performance feature for both the problem variations.

The robustness constrained problem variation presented five immediate and five pseudo-batch mode heuristics. For the immediate mode heuristics, FRMET gave the lowest makespan for both the heterogeneity trials, but it also had a high number of failed trials (for HIHI heterogeneity). The FRKPB heuristic had the lowest number of failed trials for HIHI heterogeneity. For the pseudo-batch mode, FMCTMCT performed the best in terms of both the makespan and failed number of trials. The immediate mode heuristics described here can be used when the individual guarantee for the submitted jobs is to be maintained (as there is no reordering of the submitted jobs), while the pseudo-batch heuristics can be used when the overall system performance is of importance.

For the makespan constrained problem variation five pseudo-batch heuristics were designed and evaluated. A process of fine tuning was also adapted to maximize the robustness level. Of the proposed heuristics, MxMxMnMn with fine tuning performed the best for the proposed simulation environment.

CHAPTER 5

MEASURING THE ROBUSTNESS OF RESOURCE ALLOCATIONS IN A STOCHASTIC ENVIRONMENT

5.1 Introduction and Problem Statement

Often, parallel and distributed computing systems must operate in an environment replete with uncertainty while providing a required level of quality of service (QoS). This reality has inspired an increasing interest in robust design. The following are some examples. The Robust Network Infrastructures Group at the Computer Science and Artificial Intelligence Laboratory at MIT takes the position that "... a key challenge is to ensure that the network can be robust in the face of failures, time-varying load, and various errors." The research at the User-Centered Robust Mobile Computing Project at Stanford "concerns the hardening of the network and software infrastructure to make it highly robust." The Workshop on Large-Scale Engineering Networks: Robustness, Verifiability, and Convergence (2002) concluded that the "Issues are ... being able to quantify and design for robustness ..." There are many other projects of similar nature at other schools and organizations.

To provide insight into the target systems operating under uncertainty that must maintain a certain level of QoS, consider the following two examples.

Fig. 15 schematically depicts part of a total ship computing environment in the Adaptive and Reflective Middleware Systems (ARMS) program supported by DARPA's Information Exploitation Office [10]. This part of the ARMS example represents a large class of systems that operate on *periodically updated* data sets, e.g., surveillance for homeland security, monitoring vital signs of medical patients. Typically, in such systems, sensors (e.g., radar, sonar, video camera) produce data sets with a constant period of Λ time units. Periodic

The research presented in this chapter was jointly conducted with my colleague Vladimir Shestak [86,89].

data updates imply that the total processing time for any given data set must not exceed Λ, i.e., Λ is an imposed timing QoS constraint for the system. Suppose that each input data set must be processed by a collection of N independent applications that can be executed in parallel on the available set of M heterogeneous compute nodes. Due to the changing physical world, the periodic data sets produced by the system sensors typically vary in such parameters as the number of observed objects present in the radar scan and signal-to-noise ratio. Variability in the data sets results in variability in the execution times of processing applications. Due to an inability to precisely predict application execution times, they can be considered uncertainty parameters in the system.

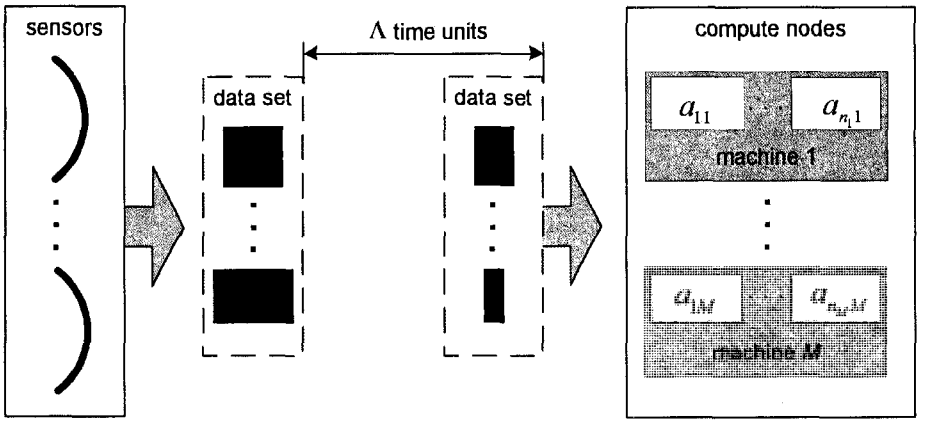


Figure 15: Major functional units and data flow for a class of system that operates on periodically updated data sets. The a_{ij}'s denote applications executing on machine j. Processing of each data set must be completed within Λ time units.

An important task for a mapper (resource management system) is to distribute processing applications across compute nodes such that a produced resource allocation is *robust*, i.e., it can guarantee (or has a high probability) that the imposed QoS constraint is satisfied despite uncertainties in application execution times.

Another example of a distributed computing system that must accommodate uncertainty under tight timing QoS constraints is a web search engine. In the Google search engine [16], the user query response time is required to be at most 0.5 seconds—including network round trip communication latency. Query execution in this system consists of two major phases. The first phase produces an ordered list of document identifiers (docids). This list is a result

of merging the responses from multiple index servers, each searching over a particular subset (index shard) of the entire index database. The second phase uses the list of docids and computes the actual title and uniform resource locators of these documents, along with any query-specific document summary information. Document servers perform this job, each processing a certain part of the docids list.

Consider the first phase of the system where a fork-join job [59] must be performed, as shown in Fig. 16 (similar analysis can be derived for phase 2). To speed up overall execution time, each query is split into multiple copies which are processed in parallel by a subset of the available index servers—chosen by the cluster manager such that they cover the entire index database. Each copy queues to a different index server, and each index server has its own input buffer where it serves requests in the order of their arrival (for simplicity of analysis, sequential query processing at each index server is considered in this study). The cluster manager must be able to accommodate uncertainty in query processing times because the exact time required to process a query is not known *a priori*. However, it is possible for the fork node to use the attributes of an incoming query to identify a subset of the past queries that have similar attributes and share a common distribution of execution times. These past execution times taken from the identified subset of queries can be used to create a probability density function (pdf) that describes the possible execution times for the incoming query.

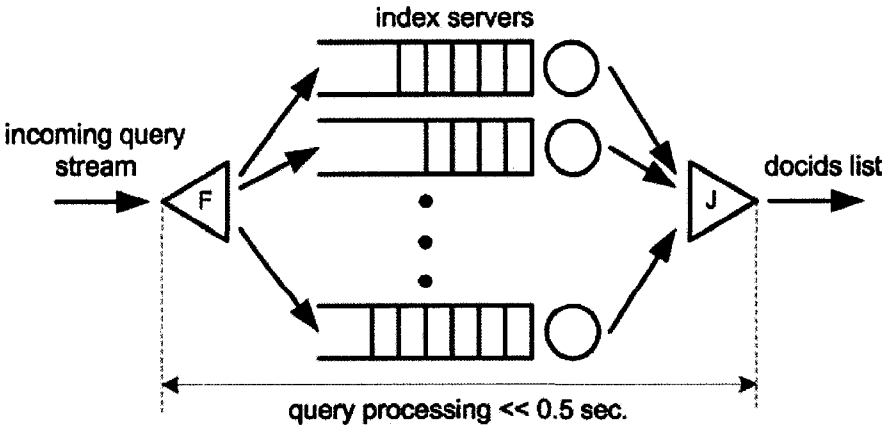


Figure 16: Fork (F) and Join (J) query processing in the first phase of Google Web search engine.

In both examples, simple load balancing algorithms may be sufficient when a distributed system is not over-subscribed, i.e., the number of queued tasks at each compute node is small. However, more sophisticated stochastic analysis is required for resource allocation as the system experiences workload surges or a loss of resources.

Robust design for such systems involves determining a resource allocation that can account for uncertainty in a way that enables the system to provide a probabilistic guarantee that a given QoS is achieved. We define a stochastic methodology for quantifiably determining a resource allocation's ability to satisfy QoS constraints in the midst of uncertainty in system parameters.

In this work, a new stochastic robustness metric is presented where the uncertainty in system parameters and its impact on system performance are modeled stochastically. This stochastic model is then used to derive a quantitative evaluation of the robustness of a *given* resource allocation, which is interpretable as the likelihood that the resource allocation will satisfy the expressed QoS constraints. The problem of deriving a resource allocation represents a large body of research in the field of parallel and distributed computing (e.g., [1, 26, 32, 36, 39, 42, 50, 61, 66, 86, 103]), In this chapter, we will analyze the utility of the proposed stochastic robustness metric by applying the metric to resource allocations in a simulated prototype of the distributed system described in the ARMS Example. The simulation results of the application are also compared with a deterministic approach for determining the robustness of a resource allocation. In Chapter 6, we will utilize the derived stochastic robustness metric to design iterative resource allocation heuristics that leverage the metric to perform resource allocation.

The major contribution of this work is a mathematical model for a stochastic robustness metric that utilizes the available information to quantifiably determine a resource allocation's ability to satisfy expressed QoS constraints. In addition, the utility of the proposed metric is demonstrated by comparison with a common performance measure of resource allocations, as well as a comparison with a similar metric taken from the literature. We show that when the additional information required by the stochastic model is available, a better selection among resource allocations is possible. Further, this work presents two

alternative means for computing the metric that render the required computation practical in a number of common environments.

The remainder of this chapter is organized in the following manner. Section 5.2 develops the general framework upon which the stochastic robustness metric is built. Specifically, in Subsection 5.2.1 a formal definition of stochastic robustness is given. Subsection 5.2.2 discusses methods of computing the stochastic robustness metric given the independence of input parameters. A bootstrap method for estimating probabilities in distributed systems is analyzed in Subsection 5.2.3. A numerical study is included in Section 5.3 to further validate the utility of the proposed methodology. Section 5.4 presents this work in relation to the published work from the literature. Section 5.5 concludes the paper.

5.2 Mathematical Model of Stochastic Robustness

5.2.1 Definition of Stochastic Robustness

The derivation of a stochastic robustness metric for a given distributed computing environment requires a mathematical model that accounts for the existing uncertainty to reasonably predict the performance of the system. To emphasize the distinction between the system and its mathematical model, any new terminology related to the model will explicitly reference the model.

In the ARMS example, let $\underline{S_j}$ be the sequence of $\underline{n_j}$ applications assigned to compute node j in the order they are to be executed, i.e., $S_j = [a_{1j}, a_{2j}, ..., a_{n_j j}]$. In the Google example, sequence S_j represents n_j queries assigned to index server j. Let random variable $\underline{T_{ij}}$ denote the execution time of each individual application (query) $\underline{a_{ij}}$ on compute node (index server) j. In a variety of systems, the execution time T_{ij} represents the time required for a_{ij} to process an individual data set on compute node j. The random variables T_{ij} serve as the inputs to the mathematical model that characterize the uncertainty in execution time for each of the applications in the system. These random variables will be referred to as the uncertainty parameters of the mathematical model. The performance of the considered distributed system is measured according to an established performance metric and may be different on a per system basis [1,57]. In the mathematical model, the system performance

ψ referred to as the performance characteristic, is an output of the mathematical model of the system.

In the ARMS example, the evaluation of system performance is based on the makespan value (total time required for all applications to process a given data set) [26] achieved by a given resource allocation, i.e., a smaller makespan equates to better performance. The functional dependence between the uncertainty parameters and the performance characteristic in the model can be expressed mathematically as

$$\psi = \max\{\sum_{i=1}^{n_1} T_{i1}, ..., \sum_{i=1}^{n_M} T_{iM}\}. \tag{36}$$

In the Google example, the performance in phase 1 is measured for each individual query. Unlike the ARMS example where the evaluation of makespan values occurs at each Λ prior to the execution of any application, query performance evaluation in the Google example is performed while the system is busy processing queries. Assume that M copies of a query arrive at index servers at wall-clock time t, and n_j is the number of queries pending execution or being executed by index server j at that time. Let t_{0j} denote the wall-clock start time of execution for the query being processed by index server j at time t. In the corresponding mathematical model, the functional dependence between the uncertainty parameters and the performance characteristic at time t, denoted as $\psi(t)$, can be stated as

$$\psi(t) = \max\{T_{11} - (t - t_{01}) + \sum_{i=2}^{n_1} T_{i1}, ..., T_{1M} - (t - t_{0M}) + \sum_{i=2}^{n_M} T_{iM}\}. \tag{37}$$

Due to its functional dependence on the uncertainty parameters T_{ij}, the performance characteristic ψ is itself a random variable.

Let the QoS constraints be quantitatively described by the values β_{min} and β_{max} limiting the acceptable range of possible variation in the system performance [3], i.e., $\beta_{min} \leq \psi \leq \beta_{max}$. **The stochastic robustness metric is the probability that the performance characteristic of the system is confined to the interval $[\beta_{min}, \beta_{max}]$, i.e., $\mathbb{P}[\beta_{min} \leq \psi \leq \beta_{max}]$.** For a given resource allocation, the stochastic robustness quantitatively measures the likelihood that the generated system performance will satisfy

the stipulated QoS constraints. Clearly, unity is the most desirable stochastic robustness metric value, i.e., there is zero probability that the system will violate the established QoS constraints.

5.2.2 Independence Assumption

In the model of compute node j, the functional dependence between the set of local uncertainty parameters $\{T_{ij} | 1 \leq i \leq n_j\}$ and the local performance characteristic ψ_j can be stated in the ARMS example as $\psi_j = \sum_{i=1}^{n_j} T_{ij}$; in the Google example as $\psi_j = T_{1j} - (t - t_{0j}) + \sum_{i=2}^{n_j} T_{ij}$.

Independence of the local performance characteristics implies that the random variables $\psi_1, \psi_2, ..., \psi_M$ are mutually independent. If such independence is established, the stochastic robustness in a distributed system can be expressed as the product of the probabilities of each compute node meeting the imposed QoS constraints. Mathematically, this is given as

$$\mathbb{P}[\beta_{min} \leq \psi \leq \beta_{max}] = \prod_{j=1}^{M} \mathbb{P}[\beta_{min} \leq \psi_j \leq \beta_{max}]. \tag{38}$$

Specifically in (38), $\beta_{max} = \Lambda$ in the ARMS example and $\beta_{max} \ll 0.5$ sec. in the Google example. In both examples β_{min} is set to zero because there is no minimum time constraint on execution.

If the execution times T_{ij} of applications mapped on a compute node j are mutually independent (e.g., this assumption is valid for non-multitasking execution mode commonly considered in the literature [26, 36, 59, 66, 103]), then $\mathbb{P}[\beta_{min} \leq \psi \leq \beta_{max}]$ can be computed using an n_j-fold convolution of probability density functions (pdfs) $f_{T_{ij}}(t_i)$ [63]

$$\mathbb{P}[\beta_{min} \leq \psi_j \leq \beta_{max}] = \int_{\beta_{min}}^{\beta_{max}} [f_{T_{ij}}(t_1) * ... * f_{T_{n_j j}}(t_{n_j})] dt. \tag{39}$$

An n_j-fold convolution of (39) requires $n_j - 1$ computations of the convolution integral [63]; thus, a direct numerical integration may become a formidable task when n_j is a relatively large number. However, a high quality approximation to the n_j-fold convolution can be obtained, at a low computational expense, by applying Fourier transforms. Thus, if $\Phi_{T_{ij}}(\omega)$

denotes the characteristic function [78] of T_{ij}, then (39) can be computed as follows

$$\mathbb{P}[\beta_{min} \leq \psi_j \leq \beta_{max}] = \int_{\beta_{min}}^{\beta_{max}} \Phi_{\psi_j}^{-1}\{\Phi_{T_{1j}}(\omega) \times ... \times \Phi_{T_{n_j j}}(\omega)\}. \tag{40}$$

From this point on we assume that each pdf $f_{T_{ij}}(t_i)$ is expressed as a discrete probability mass function (pmf) utilizing $\underline{\Omega}$ points—this is common in practical implementations. As such, the calculation can be performed in the frequency domain using a Fast Fourier Transform (FFT) that reduces the computational cost of finding the corresponding characteristic functions $\Phi_{T_{ij}}$. The FFT method is a discrete Fourier transform algorithm that reduces the number of computations needed for Ω points from $2\Omega^2$ to $2\Omega \log \Omega$ [78]. Thus, the computational complexity of determining the local performance characteristic can be drastically reduced, making the approach reasonable to compute.

Table 12 shows the empirical computation times required to execute n-fold convolution with respect to two different levels of n and four different levels of Ω. The table demonstrates that, as the number of sequential convolution operations grows, the corresponding computation time increases at a reasonable rate. This result reflects a potential applicability of the stochastic robustness metric for a broad spectrum of distributed systems where the imposed QoS constraints are either substantially longer than the total time needed for a mapper to execute a required number of convolutions, or a mapping is generated in off-line fashion, i.e., this time is not an issue.

Table 12: Computation times (sec.) required to achieve different levels of n-fold convolution computed with the Fast Fourier Transform method.

n in n-fold convolution	number of points Ω in T_{ij}'s pmf			
	62	128	256	512
10	0.0216	0.0462	0.0953	0.2059
100	1.2	1.79	3.57	7.28

In dynamic systems, processing a continuous stream of tasks (e.g., in the Google example), the number of convolutions required at each mapping event is relatively low. For example, evaluating a potential allocation of a given task on a particular compute node

requires only one convolution of the *execution* time distribution for the task with the *completion* time distribution of the the task assigned last to the considered compute node. Once the assignment of a given task is finalized, its computed completion time distribution will be used for future assignment assessments.

5.2.3 Bootstrap Approximation

This subsection presents an alternative method of evaluating $\mathbb{P}[\beta_{min} \leq \psi_j \leq \beta_{max}]$ known in the literature as the bootstrap method [104]. In contrast to convolution that is applicable only when $\psi_j = \sum_{i=1}^{n_j} T_{ij}$, the bootstrap procedure can be applied to *various* forms of functional dependence between local uncertainty parameters T_{ij} and the local performance characteristic ψ_j, making it very useful in practical implementations. For example, processing of queries by a Web server is typically done in a parallel multitasking environment, and there exists a complex functional dependence [9] between the time required to process the query and a number of currently executing threads, amount of data cached, types of requests, etc.

Suppose that for each T_{ij}, there are k sample observations obtained as a result of past executions of application i on compute node j. As k grows, new sample observations are added, and the sample pmf $\widehat{f}_{(k)T_{ij}}(t_i)$, constructed from these observations, converges in probability to $f_{T_{ij}}(t_i)$, i.e., $\widehat{f}_{(k)T_{ij}}(t_i) \xrightarrow{\mathbb{P}} f_{T_{ij}}(t_i)$. Let $\widehat{T}^*_{(k)ij}$ denote one draw from the sample distribution $\widehat{f}_{(k)T_{ij}}(t_i)$. Let $\widehat{\psi}^*_{(k)j}$ be a bootstrap replication whose computation is based on a known functional dependence between the set of drawn $\widehat{T}^*_{ij}$ and ψ_j, i.e., $\widehat{\psi}^*_{(k)j} = g(\widehat{T}^*_{(k)1j}, ..., \widehat{T}^*_{(k)n_jj})$. In the bootstrap simulation step [104], B bootstrap replications of $\widehat{\psi}^*_{(k)j}$ are computed: $\widehat{\psi}^*_{(k)j,1}, ..., \widehat{\psi}^*_{(k)j,B}$. If $\widehat{F}_{(B)\psi_j}(t)$ represents a sample cumulative density function (cdf) of ψ_j derived from these bootstrap replications, then the probability for the local characteristic function ψ_j can be approximated as

$$\mathbb{P}[\beta_{min} \leq \psi_j \leq \beta_{max}] \approx \widehat{F}_{(B)\psi_j}(\beta_{max}) - \widehat{F}_{(B)\psi_j}(\beta_{min}). \tag{41}$$

Equation (41) assumes the existence of a monotone normalizing transformation for the ψ_j distribution, and it is based on a proof of bootstrap *percentile* confidence interval [104].

An exact normalizing transformation will rarely exist, but approximate normalizing transformations may exist—the latter causes the probability that ψ_j is in the interval $[\beta_{min}, \beta_{max}]$ to be not exactly $\widehat{F}_{(B)\psi_j}(\beta_{max}) - \widehat{F}_{(B)\psi_j}(\beta_{min})$.

The pseudocode for the bootstrap analysis is as follows:

1. $B \leftarrow$ number of bootstrap replications
2. Vboot $\leftarrow$ vector of length B
3. Vsample $\leftarrow$ vector of length n_j
4. **for**(b in $1 : B$) {
5. **for**(i in $1 : n_j$) {
6. Vsample $\leftarrow$ sample with replacement $\widehat{f}_{(b)T_{ij}}(t_i)$
7. }
8. Vboot $\leftarrow g$(Vsample)
9. nullify Vsample
10. }
11. construct $\widehat{F}_{(B)\psi_j}(t)$ from Vboot
12. Nsamples $\leftarrow$ number of samples in Vboot $\in [\beta_{min}, \beta_{max}]$
13. $\mathbb{P}[\beta_{min} \leq \psi_j \leq \beta_{max}] \approx$ Nsamples$/B$

Table 13 presents the empirical data for an experiment conducted to illustrate the accuracy of the bootstrap approximation for the case where the functional dependence between T_{ij} and ψ_j was a summation. Table 13 captures the percent error of the achieved approximations based on equation 41 with respect to the exact convolution results. In the experiment, β_{min} was set to 0, β_{max} was set to the mean value of t from $\widehat{F}_{(B)\psi_j}$, and all T_{ij} distributions were modeled by randomly assigning a probability associated with each of Ω data points with final normalizations. Each value in Table 13 represents the average

across 100 different trials. As the results show, (1) relative accuracy remains insensitive to the number of applications assigned to compute node j, (2) tighter approximations were obtained by increasing the number of bootstrap replications. If distributions of uncertainty parameters were closer to normal—which occurs often in practice—the resultant bootstrap approximations would be more precise as described in the proof of equation 41 [104]. There are other bootstrap approximations that may be more accurate, especially when the nature of the expected cdf of the performance metric is known. However, some bootstrap methods require a significant amount of computation and might be prohibitively expensive in certain distributed systems.

Table 13: Percent error achieved with bootstrap approximations.

n_j	number of bootstrap replications		
	100	1000	10000
10	5.63	5.61	2.16
100	8.35	3.23	2.13
1000	6.52	2.84	1.04

5.3 Example Application of Stochastic Robustness

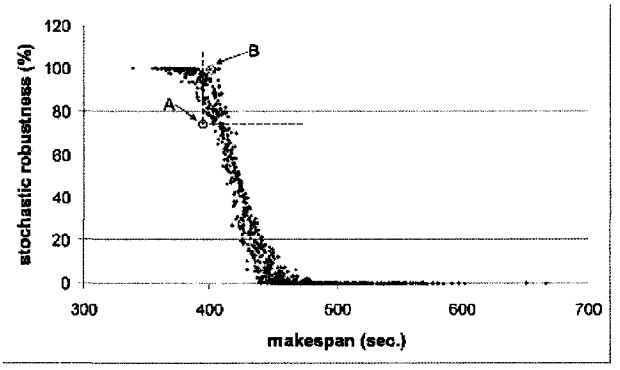
The experiments in this section seek to establish the utility of the stochastic robustness metric in distinguishing between resource allocations that perform similarly in terms of the deterministic robustness metric from [3] and a commonly used metric, such as makespan. The simulation of the system outlined in the ARMS example of Section 5.1 included 1000 randomly generated resource allocations where 128 independent applications ($N = 128$) were allocated to eight machines ($M = 8$). Each of the application execution time distributions, specific to each application-machine pair, was modeled with a discrete pdf randomly constructed on the range $[0, 40]$ seconds, inclusive. To construct each discrete pdf, ten randomly selected values spread across the range of the distribution were assigned probabilities sampled uniformly on the range $(0, 1)$. All application execution time distributions were subsequently normalized. Let $\underline{mean_{av}}$ be the average value computed across the means of all constructed application execution time distributions. In the conducted simulation, the

QoS constraint Λ was set as follows $\Lambda = 1.5 \times N \times mean_{av}/M$. Recall, for the ARMS example Λ is a QoS constraint on system processing time that is used in the definition of the stochastic robustness metric given in equation (38). In Fig. 17, the "stochastic robustness" vertical axes correspond to the probability that the makespan will be $\leq \Lambda$. In this simulation, the deterministic robustness metric and makespan were calculated using the mean of the execution time distribution for each application-machine pair in the given allocation.

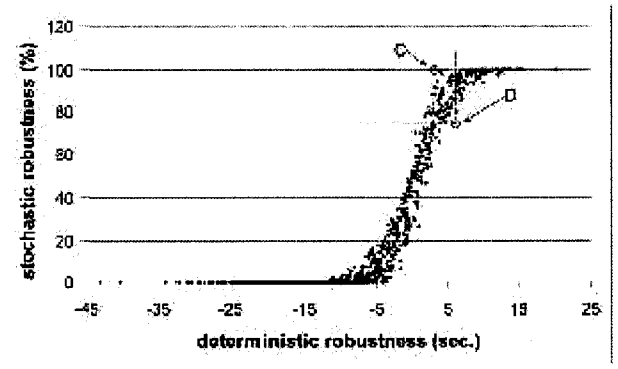
In Fig. 5.3, a comparison between the stochastic robustness metric and makespan is presented for 1000 *randomly* generated resource allocations. Fig. 5.3 demonstrates that, as can be expected, resource allocations that produce a very large makespan also tend to have a very small stochastic robustness metric value. However, as can also be seen in the figure, there can be a large discrepancy between the predicted performance as found using the expected makespan and the predicted performance using the stochastic robustness metric. For example, in the figure, compare the two resource allocations labeled A and B. If the comparison of these two resource allocations is made using the expected makespan, allocation A appears to be slightly superior to allocation B. However, resource allocation B presents a 99.8% probability of meeting the imposed QoS constraints, where as allocation A has only a 75% probability of meeting it. In this case, using only the expected makespan to compare the two resource allocations leads to a sizable increase in risk for a modest ($\approx 5\%$) improvement in the expected makespan. Any of the approximately 100 resource allocations above and to the right of allocation A, delineated by the dashed lines in the figure, will have a higher robustness value yet higher (worse) makespan value than A.

In Fig. 5.3, a comparison of the stochastic robustness metric and the deterministic robustness metric is presented for 1000 randomly generated resource allocations. In Fig. 5.3, compare the two resource allocations C and D. Based on using deterministic robustness measures as in [3], allocation D (with a deterministic measure of 6.13 sec.) is preferred over C (with a deterministic measure of 3.25 sec.). However, under the new stochastic model, allocation C (with a stochastic measure of 99.9%) is preferred over D (with a stochastic measure of 75%). Thus in this case, using only the deterministic robustness metric to select a resource allocation, D appears to be more robust than C. In contrast, the stochastic

robustness metric, which accounts for the distribution of makespan outcomes, shows that allocation C has a 99.9% probability of meeting the QoS constraint while allocation D has only a 75% probability of meeting the QoS constraint.



(a) width=5in



(b) width=5in

Figure 17: A plot of stochastic robustness metric versus (a) makespan and (b) deterministic robustness, for 1000 randomly generated resource allocations. The stochastic robustness metric values for allocations A and B exemplify the conflict between the stochastic robustness metric and makespan. Similarly, the stochastic robustness metric values for allocations C and D exemplify the conflict with the deterministic robustness metric.

Consider the sub-region identified in Fig. 5.3 with dotted lines originating from the point D, containing all of the points above and to the left of D. Each of the identified points in the sub-region has a higher stochastic robustness metric value than D but a lower deterministic robustness metric value than D.

The results also show a number of resource allocations that have a *negative* deterministic robustness value. For the data used in this simulation study, a negative value for the deterministic robustness correlates with a low stochastic robustness value.

It is shown in [3] that the deterministic robustness metric provides better information than just makespan. However, when execution time distributions are available, the stochastic robustness metric is even better.

Differences between the stochastic robustness metric and the deterministic robustness metric can be explained by the fact that the stochastic robustness metric uses information about the distribution of outcomes for the resource allocation to determine robustness. In contrast, the deterministic robustness metric uses a scalar estimate of each application's execution time on each machine to determine a resource allocation's robustness. In this study, there were a significant number of resource allocations where the stochastic robustness metric's use of the distribution of outcomes caused the metric to produce a robustness value for the allocation that failed to correlate well with the deterministic robustness metric. Thus, *if* the information needed for using the stochastic model is available, or can be obtained, *then* a better selection among resource allocations is possible.

5.4 Related Work

Prior work [3] in this area has referred to a resource allocation's tolerance to uncertainty as the robustness of that resource allocation. That work also defines a set of criteria for definitively claiming that a resource allocation is robust given a deterministic estimate for each considered system parameter. This determination of robustness begins by asking the claimant to define the behavior of the system that makes it robust, i.e., differentiate between acceptable performance and unacceptable performance of the system. Given this definition of acceptable performance, the uncertainty in system parameters must be identified along with its impact on the system's ability to deliver acceptable performance.

In [3], a four-step procedure is defined for deriving a deterministic robustness metric. The authors proposed procedure was used here to motivate the derivation of a stochastic robustness metric. According to [3], the first step in defining a robustness metric requires quantitatively describing what makes the system robust. This description establishes the required QoS level that must be delivered to refer to the system as robust—essentially bounding the acceptable variation in system performance. A pair of values, β_{min} and

β_{max} that bound each performance feature must be identified, quantitatively defining the tolerable variation in each of the performance features.

In the second step, all modeled system and environmental parameters that may impact the system's ability to deliver acceptable QoS are identified. These parameters are referred to as the perturbation parameters of the system. In our new stochastic approach, each perturbation parameter, or uncertainty parameter, is modeled as a random variable fully described with a pmf. In this way, all possible values of the considered perturbation parameters, and their associated probabilities, are included in the calculation of the stochastic robustness metric. Our new approach differs from that in [3], where a single deterministic estimated value for each of the identified perturbation parameters is used.

In the third step, the impact of the identified perturbation parameters on the system's performance features is defined. This requires identifying a function that maps a given vector of perturbation parameters to a value for the performance feature of the system. Similarly in our new stochastic environment, this involves defining the functional dependence between the input random variables and the given performance feature. However, in our new model this involves more complex computations to combine random variables.

Finally, in the fourth step, the previously identified relation is evaluated to quantify the robustness. As a measure of robustness, the authors in [3] use the "minimum robustness radius" that relies on a deterministic performance characteristic. Furthermore, it assumes there is no *a priori* information available about the relative likelihood or magnitude of change for each perturbation parameter. Thus, the minimum robustness radius is used in a deterministic worst-case analysis. In our new stochastic model, more information regarding the variation in the perturbation parameters is assumed known. Representing the uncertainty parameters of the system as stochastic variables enables the robustness metric in the stochastic model to account for all possible outcomes for the performance of the system. This added knowledge comes at a computational cost. The stochastic robustness metric requires more information and is far more complex to calculate than its deterministic counterpart. To handle the computational complexity, we considered an approximation scheme that greatly simplifies the required calculations.

In [25], the robustness of a resource allocation is defined in terms of the schedule's ability to tolerate an increase in application execution time without increasing the total execution time of the resource allocation. In this formulation, the authors define a resource allocation's robustness in terms of system slack thereby focusing their metric on a single very important uncertainty parameter, i.e., variations in application execution times. Our stochastic robustness metric is more generally applicable, allowing for any definition of QoS and able to incorporate any identified uncertainty parameters.

Our presented methodology relies heavily on an ability to model the uncertainty parameters as stochastic variables. Several previous efforts have established a variety of techniques for modeling the stochastic behavior of application execution times [18,35,64]. In [18], three methods for obtaining probability distributions for task execution times are presented. The authors also present a means for combining stochastic task representations to determine task completion time distributions. Our work leverages this method of combining independent task execution time distributions and extends it by defining a means for measuring the robustness of a resource allocation against an expressed set of QoS constraints under uncertainty.

In [51], a statistical algorithm for predicting task execution times is presented. The authors present a methodology for defining data driven estimates of uncertainty parameters in a heterogeneous computing environment. In that work, the method is applied to the problem of generating an application execution time prediction given a set of observations of that application's execution times. Their model defines an application execution time random variable as the combination of two elements. The first element corresponds to a vector of known factors that have an impact on the execution time of the application and is considered to be a deterministic component of the execution time random variable. A second element accounts for all unmodeled factors that may impact the execution time of an application and represents the stochastic component of the execution time approximation. This method for predicting application execution times can be used to determine probability density functions describing the input random variables in our framework.

In [36], the authors present a derivation of the makespan problem that relies on a

stochastic representation of task execution times. The authors also demonstrate that their presented stochastic approach to scheduling can significantly reduce the actual simulated system makespan as compared to some well known scheduling heuristics that are founded in a deterministic approach to modeling task execution times. The heuristics presented in that study were adapted to the stochastic model and used to minimize the expected system makespan given a stochastic model of task execution times. In our research, the emphasis is on quantitatively comparing one resource allocation to another by deriving a metric for the resource allocation's robustness, i.e., the probability to deliver on expressed QoS constraints. Thus, [36] is focused on designing a heuristic for the makespan problem in a stochastic environment, while this paper is focused on the evaluation of the robustness of a resource allocation given a modeled stochastic environment.

5.5 Conclusion

This paper presents a stochastic robustness metric suitable for evaluating the likelihood that a resource allocation will perform acceptably, i.e., satisfy identified QoS constraints, in an uncertain environment. In addition to the general statement of the stochastic robustness metric, the derivation, mathematical description, and computational methods of the stochastic robustness metric were also presented. The stochastic robustness metric was then applied to an example class of systems operating with periodic data sets to demonstrate its utility in evaluating the robustness of a resource allocation.

Given the raw volume of computation required to evaluate such a stochastic metric, a developed approximation scheme based on the bootstrap technique and the FFT method were tested to aid the practitioner in the actual application of the metric in different real world scenarios. A conducted simulation study demonstrates the accuracy of the bootstrap approximation and a baseline timing analysis for FFT.

There are many ways that this research on stochastic robustness may be built upon in future work. The results of this work can be leveraged to develop methods for calculating the stochastic robustness metric given system parameters that include dependencies, as was discussed earlier. Another extension of this research, discussed in Chapter 5 involves

applying the stochastic robustness metric to resource allocations in a dynamic environment, where the mix of tasks to be executed is not known in advance and system feedback about completed tasks is available [93]. In the next chapter we will discuss applying this research to the design of resource allocation techniques that utilize the stochastic robustness metric to generate allocations that are more robust [85, 87].

CHAPTER 6

ITERATIVE ALGORITHMS FOR STOCHASTICALLY ROBUST STATIC RESOURCE ALLOCATION IN PERIODIC SENSOR DRIVEN CLUSTERS

6.1 Introduction

This chapter investigates the problem of robust resource allocation for a large class of heterogeneous cluster (HC) systems operating on *periodically updated* data sets. The application domains include surveillance for homeland security, monitoring vital signs of medical patients, and automatic target recognition systems. We introduced this class of sensor driven system in the previous chapter. Recall that sensors (e.g., radar, sonar, video camera) in this environment produce data sets with a constant period of Λ time units. Due to the changing physical world, these periodic data sets typically vary in such parameters as the number of observed objects present in the radar scan and signal-to-noise ratio. Suppose that each input data set must be processed by a collection of N independent applications that can be executed in parallel in an HC system composed of M compute nodes. Unpredictable changes in input data sets result in variability in the execution times of data processing applications. This makes the problem of resource allocation (i.e., allocation of resources to applications) rather challenging, especially in situations where the system experiences workload surges or loss of hardware resources, as the total processing time for a specified *percentage* of data sets must not exceed Λ. *Robust* design for such systems involves determining a resource allocation that can account for uncertainty in application execution times in a way that enables a probabilistic guarantee for a given percentage. Furthermore, in many systems of the considered class, it is highly desirable to *minimize* the period Λ

The research presented in this chapter was jointly conducted with my colleague Vladimir Shestak [85, 86, 89].

between subsequent data updates, e.g., more frequent radar scans are needed to identify an approaching target in military applications.

The major contribution of this chapter is the design of resource allocation techniques based on *iterative* algorithms [73] to address the problem of minimizing Λ while providing a probabilistic guarantee that the time required for processing a complete data set is less than or equal to Λ. The resource allocation problem has been shown, in general, to be NP-complete in HC systems [32, 44, 106]. Thus, the development of heuristic techniques to find near-optimal solutions is an active area of research, e.g., [26,50,66,73,84]. The notion of stochastic robustness, considered as a constraint in the addressed optimization problem, was established in [86] based on a developed mathematical model of this class of HC systems. Three *greedy* approaches to the resource allocation problem were then designed presented in [87]. The latter study revealed that the adopted greedy techniques required a significant amount of time to produce a resource allocation while operating in the stochastic domain. As the research continued, more sophisticated iterative algorithms were created resulting in a significant performance improvement. These techniques are the focus of this chapter.

The remainder of this work is organized in the following manner. Three iterative algorithms designed for this environment are described in Section 6.3, each determining the lowest achievable value of period Λ for the required level of stochastic robustness. The parameters of the simulation setup are discussed in Section 6.5 along with the simulation results and evaluation of the heuristics' performance. A sampling of some related work is presented in Section 6.6. Section 6.7 concludes the paper.

6.2 Simulation Setup

This research assumes that an acceptable level of stochastic robustness is specified for the considered system. Thus, the performance goal for the mapper is to find resource allocations for a given set of N applications on M compute nodes that allows for the minimum period Λ between sequential data sets while maintaining a given level of stochastic robustness θ.

To evaluate the performance of the iterative heuristics described in Section 6.3 for the considered class of HC systems operating on periodic data, the following approach was used

to simulate a cluster-based radar system. The execution time distributions for twenty eight different types of possible radar ray processing algorithms on eight ($M = 8$) heterogeneous compute nodes were generated by combining experimental data and benchmark results. The experimental data, represented by two execution time sample pmfs, were obtained by conducting experiments on the Colorado MA1 radar [53]. These sample pmfs contain times taken to process 500 radar rays of different complexity by the Pulse-Pair & Attenuation Correction algorithm [22] and by the Random Phase & Attenuation Correction algorithm [22], both executed in non-multitasking mode on the Sun Microsystems Sun Fire V20z workstation. To simulate the effect of executing these algorithms on different platforms, each sample pmf was scaled by a factor corresponding to the performance ratio of a Sun Microsystems Sun Fire V20z to each of eight selected compute nodes [1] based on the results of the fourteen floating point benchmarks from the CFP2000 suite [98]. Combining the results available from CFP2000 for fourteen different benchmarks on eight selected compute nodes and two sample pmfs provided a means for generating a 28×8 matrix where the yj^{th} element corresponds to the execution time distribution of a possible ray processing algorithm of type y on compute node m_j.

A set of 128 applications ($N = 128$) was formed for each of 50 simulation trials, where for each trial the type of each application was determined by randomly sampling integers in the range $[1, 28]$. The 50 simulation trials provide good estimates of the mean and 95% confidence interval computed for every resource allocation algorithm.

6.3 Iterative Resource Allocation Techniques

6.3.1 Overview

Three iterative algorithms were designed for the problem of finding a resource allocation with respect to the performance goal the previous section. Iterative algorithms are probabilistic search techniques that have been widely used as a tool in optimization [73, 84, 105], artificial intelligence [49], and many other areas. The first two of the developed heuristics

[1]The eight compute nodes selected to be modeled were: Altos R510, Dell PowerEdge 7150, Dell PowerEdge 2800, Fujitsu PRIMEPOWER 650, HP Workstation i2000, HP ProLiant ML370 G4, Sun Fire V65x, and Sun Fire X4100.

operate with a set of resource allocations; whereas the third heuristic iteratively changes a single resource allocation. As opposed to greedy algorithms investigated in our previous work, where a single complete resource allocation was "constructed" [87], iterative heuristics progress toward a final solution through modified versions of complete resource allocations. In each iteration, the existing complete resource allocation (or resource allocations) is modified and evaluated. Such an iterative search process continues until an appropriate stopping criterion is reached.

To establish a basis for the comparison of the developed iterative algorithms and demonstrate the performance over time for each of them, a unique stopping criterion (USC) of 150,000 evaluations of the produced resource allocations was used in this study. It is important to note that the evaluation in the considered stochastic domain is the most computationally intensive part of any of the developed algorithms as it calls for M executions of $(n_j - 1)$-fold convolutions, followed by a recursive search for a minimum Λ level. The evaluation mechanism, referred to as the Period Minimization Routine, is described next.

Period Minimization Routine: The PMR procedure determines the minimum possible value of Λ for a *given* resource allocation and a *given* level of stochastic robustness. As a first step, the results of $(n_j - 1)$-fold convolutions are obtained with the FFT or bootstrap methods for each compute node corresponding to the completion time (i.e., $\sum_{i=1}^{n_j} T_{ij}$) distributions expressed in a pmf form. The completion time pmf for compute node m_j is comprised of K_j impulses, where every impulse corresponds to a possible pair of time outcome t_{kj} and associated probability p_{kj} for $k \in [1, K_j]$.

As a second step, the minimum Λ is determined recursively as the smallest value among $\{t_{kj} \mid 1 \leq k \leq K_j, 1 \leq j \leq M\}$, such that the specified level of stochastic robustness is less than or equal to $\prod_{j=1}^{M} \sum_{k=1}^{K_j} (p_{kj} \times \mathbf{1}(t_{kj} \leq \Lambda))$, where $\mathbf{1}(condition)$ is 1 if *condition* is true; 0 otherwise. The PMR procedure is summarized in Fig. 18.

After Ω iterations, the PMR procedure reduces the uncertainty range by the factor $\approx (0.5)^{\Omega}$, which is the fastest possible uncertainty reduction rate. This optimality becomes possible due to the fact that θ is strictly increasing as the number of impulses considered for its computation grows.

$lo = t_1 \leftarrow \min\{t_{kj} \mid 1 \leq k \leq K_j, 1 \leq j \leq M\}$;
$hi = t_2 \leftarrow \max\{t_{kj} \mid 1 \leq k \leq K_j, 1 \leq j \leq M\}$;
$P \leftarrow$ specified level of $\mathbb{P}[\psi \leq \Lambda]$;
while $\exists\ t_{kj} \in (lo, hi) \mid \{1 \leq k \leq K_j, 1 \leq j \leq M\}$

$$\mathbb{P}[\psi \leq \Lambda] \leftarrow \prod_{j=1}^{M} \left[\sum_{k=1}^{K_j} p_{kj} \times \mathbf{1}(t_{kj} \in [t_1, t_2]) \right];$$

switch $\mathbb{P}[\psi \leq \Lambda]$:

$= P$: **return**;
$> P$: $hi \leftarrow t_2$;
$< P$: $lo \leftarrow t_2$;

end of switch

$t_2 \leftarrow t_{kj} \mid \{1 \leq k \leq K_j, 1 \leq j \leq M\}$ closest to $lo + (hi - lo)/2$;
end of while
$\Lambda \leftarrow hi$.

Figure 18: The Period Minimization Routine procedure

6.3.2 Steady State Genetic Algorithm

The adopted genetic algorithm (GA) implementation was motivated by the Genitor evolutionary heuristic [105]. For the considered system, each chromosome in the GA models a complete resource allocation as a vector of numbers of length N where the i^{th} element of the vector identifies the compute node assignment for application a_i. Order in which applications are placed in a chromosome does not play any role and can be considered arbitrary. The population size for the GA was fixed at 200 for each iteration. The population size was chosen experimentally by varying the population size between 100 and 250 in increments of 50. For the samples tried, a value of 200 performed the best and was chosen for all trials. The initial members of the population were generated by applying the one-phase sorting greedy heuristic presented in [87], in which the application ordering used as an input was perturbed to produce different resource allocations.

The GA used in this work was implemented as a *steady state* GA, i.e., for each iteration of the GA only a single pair of chromosomes will be selected for crossover. Selection for crossover was implemented as rank based selection where the population of chromosomes is

sorted according to their evaluation of Λ values. For the considered problem, the most fit chromosome corresponds to a resource allocation with the smallest Λ value supportable at the specified level of stochastic robustness θ. Each chromosome generated by crossover or mutation is inserted into the population according to its evaluation such that after insertion the population remains sorted according to each chromosome's evaluation. After insertion the population is truncated to the original population size.

To reduce the number of duplicate chromosome evaluations, each chromosome that is trimmed from the active population is recorded in a list of known bad chromosomes referred to as the graveyard. Selecting the size of the graveyard reflected a trade-off between the time required to identify that a new chromosome was not present in the population and the time required to evaluate the new chromosome.

To maintain the selective pressure of rank based selection an additional constraint was placed on the population where each chromosome in the population must be unique, i.e., clones are explicitly disallowed. If a chromosome produced by the crossover operation were to generate a clone of an individual already present in the population or the graveyard, then that clone would be discarded prior to its evaluation.

The crossover operator was implemented using a two-point reduced surrogate procedure [105] where the elements between the crossover points are exchanged between the two parents. In the reduced surrogate approach, crossover points are selected such that at least one element of the parent chromosomes differs between the selected crossover points. By selecting the crossover points in this way, each child chromosome produced by the crossover operation is guaranteed to not be a clone of its parents. However, the chromosome may still be a clone of another member of the population or the graveyard. Therefore, prior to evaluating each child chromosome the GA first attempts to locate the chromosome in either the population or the graveyard. Only if the chromosome is not present in either will it be evaluated and inserted into the population.

The final step in a single iteration of the GA is mutation. For each iteration of the GA, the mutation operator is applied to every element of each chromosome in the population with probability referred to as the mutation rate. For the simulated environment, the best

results were achieved using a mutation rate of 0.01. For a given application, the mutation operator randomly selects a different compute node assignment from a subset of compute nodes. The subset includes compute nodes providing the smallest mean values of execution time distributions for a given application. The best results in the simulation studies were achieved when the size of this subset was set to three.

The steady state GA procedure is summarized in Fig. 19.

```
generate initial population;
evaluate each chromosome;
rank population based on Λ values;
while USC not met
   select two chromosomes from the population;
   select crossover points;
   exchange compute node assignments
   between crossover points;
   ascertain if either offspring are unique;
   insert unique offspring into population;
   trim population down to population size
   move dead chromosomes to the graveyard;
   for each element of each chromosome in population
     generate a random number x in the range [0,1];
     if x < mutation rate
        arbitrarily change the compute node assignment
        of the selected application;
   ascertain if the mutated chromosome is unique;
   insert unique offspring into population;
   trim population down to population size;
   move dead chromosomes to the graveyard;
end of while
output the best solution.
```

Figure 19: The steady state Genetic Algorithm procedure

6.3.3 Ant Colony Optimization

The ACO heuristic belongs to a class of swarm optimization algorithms where low-level interactions between artificial (i.e., simulated) ants result in large scale optimizations by the larger ant colony. The technique was inspired by colonies of real ants that deposit a chemical substance (pheromone) when searching for food. This substance influences the behavior of individual ants. The greater the amount of pheromone on a particular path, the larger the probability that an ant will select that path. Artificial ants in ACO behave

in a similar manner by recording their chosen path in a global pheromone table.

The ACO algorithm implemented here is a variation of the ACO algorithm design described in [37]. During ACO execution, the $N \times M$ pheromone table is maintained and updated allowing the ants to share global information about good compute nodes for each application. Let $\tau(a_i, m_j)$ represents the "goodness" of compute node m_j for application a_i. At a high level, the ACO heuristic works in the following way. A certain number of ants are released to find different complete mapping solutions. Based on the mapping produced by the individual ants, the pheromone table is updated. This procedure is repeated as long as the unique stopping criterion is not reached. The final mapping solution is determined by mapping each application to its highest pheromone value compute node.

At a low level, each ant heuristically "constructs" its complete mapping, and its mapping decision process balances between the (a) performance metric and (b) pheromone table information. The ant procedure involves two phases. In Phase 1, for each unmapped application, the compute node, denoted as $m_{best}(a_i)$, is determined such that it would provide the minimum, $\mu_{min}(a_i)$, across M compute nodes with respect to the mean values of the completion time distributions, obtained as a mapping with a_i added was evaluated for each of these compute nodes. The worth of application a_i, denoted as $\eta(a_i)$, is then determined as a result of the following normalization

$$\eta(a_i) = \frac{\mu_{min}(a_i)}{\sum_{\text{unmapped } a_k} \mu_{min}(a_k)}. \tag{42}$$

In Phase 1, instead of operating with mean values, intermediate minimum levels of Λ could be computed through PMR calls. However due to the time consuming FFT executions, this approach significantly slows down each ant's production of a completed resource allocation which, in turn, limits the number of high level iterations.

In Phase 2, an unmapped application is stochastically selected (procedure described later) and assigned to its $m_{best}(a_i)$ compute node. The ant procedure is repeated until all applications are mapped.

Let the fitness of ant s, denoted as $f(s) \in (0, 1)$, be determined as a rank of ant s in the

sorted order of ants in the current iteration. Sorting is based on the minimum possible level of Λ, obtained as a PMR call invoked at the end of each ant procedure. The pheromone table is updated at the end of each high level iteration, i.e., when all ants complete their paths. Specifically, if ρ denotes a coefficient that represents pheromone evaporation, B_s denotes the set of application-compute node assignments comprising the path of ant s, and assuming Q ants released, each $\tau(a_i, m_j)$ is updated as follows

$$\tau(a_i, m_j) = \rho \times \tau(a_i, m_j) \; + \sum_{s=1}^{Q} \frac{f(s)}{N} \times \mathbf{1}(a_i \text{ assigned to } m_j \text{ in } B_s). \tag{43}$$

Initially, all values in the pheromone table were set to 1.

Let α be the scalar that controls the balance between the pheromone value and worth. The probability that ant s selects application a_i to be mapped next is

$$\mathbb{P}[a_i \text{ selected next}] = \frac{\alpha \times \tau(a_i, m_{best}(a_i)) + (1-\alpha) \times \eta(a_i)}{\sum\limits_{\text{unmapped } a_k} \alpha \times \tau(a_k, m_{best}(a_k)) + (1-\alpha) \times \eta(a_k)}. \tag{44}$$

The scalar α was determined experimentally by incrementing from 0 to 1 in 0.1 steps. In the simulation trials tested, the performance peak was detected with α equal to 0.5. The pheromone evaporation factor ρ of 0.01 was determined in a similar manner. The total number of ants for each iteration was set to 50; any further increase of this number in the experiments resulted in performance degradation.

The ACO procedure is summarized in Fig. 20.

6.3.4 Simulated Annealing

The SA algorithm—also known in the literature as Monte Carlo annealing or probabilistic hill-climbing [73]—is based on an analogy taken from thermodynamics. In SA, a randomly generated solution, structured as the chromosome for GA and used as a startup point, is then iteratively modified and refined. Thus, SA in general, can be considered as an iterative technique that operates with one possible solution (i.e., resource allocation) at a time.

To deviate from the current solution in an attempt to find a better one, SA repetitively performs the mutation operation in the same fashion as for GA. Once a new *unique* solution,

```
initialize pheromone table;
while USC not met
   for each ant
      while there are unmapped applications
         select application a_i according to (44);
         map application a_i to m_best(a_i) compute node;
         break ties arbitrarily;
      end of while;
      compute f(s) via PMR call;
   update pheromone table according to (43);
end of while
map each application a_i to its m_best(a_i) compute node.
```

Figure 20: The Ant Colony Optimization procedure

denoted as S_{new} is produced (SA relies on the same GA concept of uniqueness), a decision regarding the replacement of a previous solution with a new one has to be made. If the quality of a new solution, $\Lambda(S_{new})$, found after evaluation, is higher than the old solution, the new solution replaces the old one. Otherwise, SA uses a procedure that probabilistically allows poorer solutions to be accepted during the search process, which makes this algorithm different from other strict hill-climbing algorithms [73]. This probability is based on a system temperature, denoted as $\mathbb{T}$, that decreases with each iteration. As the system temperature "cools down" it becomes more difficult for poorer solutions to be accepted. Specifically, in the latter case, the SA algorithm selects a sample from the range $[0, 1)$ according to the uniform distribution. If

$$random[0,1) > \frac{1}{1+\exp^{(\frac{\Lambda(S_{old})-\Lambda(S_{new})}{\mathbb{T}})}} \tag{45}$$

the new poorer resource allocation is accepted; otherwise, the old one is kept. As it follows from (45), the probability for a new solution of similar quality to be accepted is close to 50%. In contrast, the probability of poor solutions to be rejected is rather high, especially when the system temperature becomes relatively small.

After each mutation (described in the GA procedure) that successfully produces a new unique solution, the system temperature $\mathbb{T}$ is reduced to 90% of its current value. This percentage, defined as a cooling rate, was determined experimentally by varying the rate in

the range of $(0, 1]$ in 0.1 steps. The initial system temperature in (45) was set to the Λ of the chosen startup resource allocation.

The SA procedure is summarized in Fig. 21.

```
S_old ← initial randomly generated resource allocation;
𝕋 ← Λ(S_old);
while USC not met
   S_new ←result of successful mutation;
   if Λ(S_new) < Λ(S_old)
      S_old ← S_new;
   else if (45) holds
         S_old ← S_new;
𝕋 ← 0.9 × 𝕋;
end of while
```

Figure 21: The Simulated Annealing procedure

6.4 Lower Bound Calculation

To evaluate the absolute performance attainable by the developed resource allocation techniques, a lower bound (LB) on the minimum period Λ was derived based on the assumption that the specified level of the stochastic robustness metric is greater than or equal to 0.5, i.e., $\theta \geq 0.5$. A requirement to support levels of stochastic robustness metric below 0.5 appears to be quite unreasonable for practical implementations; therefore, it is not considered in this work. The process of calculating LB involves two major steps. In the first step, a "local" lower bound on Λ is established for a given mapping. In the second step, a unique LB is computed for all possible local lower bounds by solving a relaxed form of the Integer Linear Program formulated for the addressed resource allocation problem.

Step 1: Consider a *given* complete resource allocation of N applications on M compute nodes. Let $\overline{\Lambda}$ denote the maximum of the means across all M completion time distributions, $\mu(\sum_{i=1}^{n_j} T_{ij})$, i.e., $\overline{\Lambda} = \max\{\mu(\sum_{i=1}^{n_j} T_{ij}) \mid 1 \leq j \leq M\}$. As an assumed level of the stochastic robustness metric is greater than or equal to 0.5, $\overline{\Lambda}$ represents the smallest possible time period for a given mapping. To observe this, recall that

1. mean $\mu(a)$ is a "center of mass" of the distribution of random variable $\underline{a}$, so that if $\underline{z}$ is the compute node given by $z = \text{argmax}\{\mu(\sum_{i=1}^{n_j} T_{ij}) \mid 1 \leq j \leq M\}$, then $\mathbb{P}[\psi_z \leq \overline{\Lambda}] = 0.5$;

2. $\mathbb{P}[\psi_z \leq \overline{\Lambda}] \geq \mathbb{P}[\psi \leq \overline{\Lambda}]$ because according to (38), $\mathbb{P}[\psi \leq \overline{\Lambda}]$ is computed as an M-product of $\mathbb{P}[\psi_j \leq \overline{\Lambda}]$, where each of M terms is less than or equal to one.

Step 2: An objective here is to determine LB, denoted as $\underline{\Lambda^*}$, such that $\Lambda^* \leq \min\{\overline{\Lambda} \mid$ all possible mappings$\}$. Relying on an important property of sum of means stating that $\sum_{i=1}^{n_j} \mu(T_{ij}) = \mu(\sum_{i=1}^{n_j} T_{ij})$, the problem of finding Λ^* can be formulated in the following Integer Linear Programming (ILP) form.

Let a *binary* decision variable $\underline{x[i,j]} \mid \{1 \leq i \leq N; 1 \leq j \leq M\}$ be equal to one if application a_{ij} is assigned to compute node j, and equal to zero if a_{ij} is not assigned to compute node j. The ILP objective function can be stated as

$$\text{minimize} \quad \Lambda^* = \max\{\sum_{i=1}^{N} \mu(T_{ij}) \times x[i,j] \mid 1 \leq j \leq M\}.$$

The objective function is subject to conditions (a) and (b):

$$x[i,j] \in \{0,1\} \quad \text{for } 1 \leq i \leq N, \quad 1 \leq j \leq M; \quad \text{(a)}$$

$$\sum_{i=1}^{N} x[i,j] = 1 \quad \text{for } 1 \leq j \leq M; \quad \text{(b)}$$

In addition to condition (a) explained above, condition (b) forces each application to be mapped to the system. For small-scale problems, a global optimal solution can be found for the derived ILP form in a reasonable time (e.g., by applying Branch-and-Bound [20]). However, condition (b) makes the ILP form NP-complete [74], so that for large-scale problems an exact solution requires an unaffordable amount of time. Thus for latter cases, a Linear Programming (LP) relaxation can be applied to the ILP form which implies that condition (a) is relaxed to real numbers, i.e., $x[i,j] \in [0,1] \mid \{1 \leq i \leq N, 1 \leq j \leq M\}$. A global optimal solution to the LP form can be found in *polynomial* time [48], and it will be a lower bound for the corresponding ILP global optimal solution Λ^*. Also note that the derived LB is tighter for stochastic robustness levels closer to 0.5; this is a result of using mean values in the LB computation.

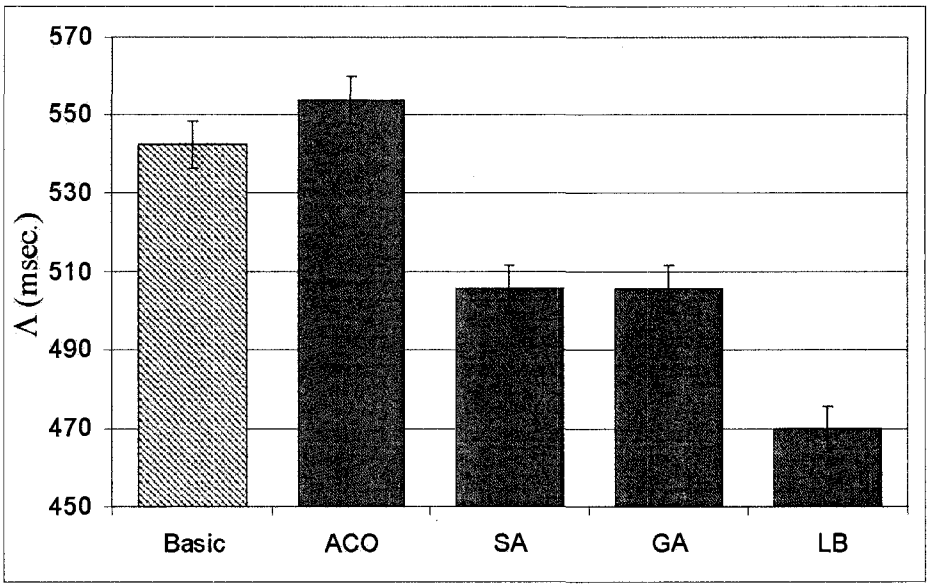


Figure 22: A comparison of the results obtained for the described heuristics where the minimum acceptable robustness value was set to be 0.90. The y-axis corresponds to a Λ value obtained by executing the corresponding heuristics. The Λ value for each heuristic corresponds to the average over 50 trials.

6.5 Experimental Results

The results of the simulation are presented in Fig. 22. Both the GA and SA heuristics were able to improve upon the results of the Basic heuristic of [87] by more than 7% with respect to the absolute performance and by 50% with respect to the derived LB. However, the ACO procedure was unable to improve upon the results of the Basic heuristic but was able to produce a result such that the confidence intervals of the ACO and Basic results were overlapping.

For the 50 trials tested, LB produced a mean minimum supportable Λ of 469.8 msec. The mean of the minimum supportable Λ for the Basic heuristic over the same 50 trials was found to be 542.5 with a 95% confidence interval of 7.07. The ACO result had a mean minimum supportable Λ value of 553.7 with a 95% confidence interval of 6.2. The SA procedure for the same trials produced a mean Λ value of 505.6 with a 95% confidence interval of 5.9. The GA result was very similar to the SA result, producing a mean Λ value of 505.3 with a 95% confidence interval of 6.1.

Both the GA and SA heuristics performed comparably in this simulation environment.

The success of the SA procedure and the near overlap of the SA and GA results may suggest that the local search procedure used in the mutation operator by both GA and SA is responsible for their marked improvement over Basic. Additional experiments were conducted with the GA without utilizing local search and although the simple GA was able to improve the average result of the Basic heuristic by almost 2% the improvement was not statistically significant. The GA with the local search mutation operator outperforms the GA without local search by a significant margin.

The ACO heuristic was unable to improve upon the results of the Basic heuristic of our previous work. This might suggest that using only the mean values of the execution time distributions to construct solutions in Phase 1 is insufficient. Instead of operating with mean values, intermediate minimum levels of Λ could be computed through PMR calls to potentially improve the results of the ACO procedure. However, this would dramatically increase the number of evaluations required by ACO to produce the ants of each iteration. In so doing, the number of high-level iterations that the ACO procedure would be able to complete within the CSC would be significantly reduced. The major hindrance to the effectiveness of ACO in this environment is that it relies on the repetitive application of a constructive heuristic within an iteration to update the pheromone table. As shown in [87], constructive heuristics such as the Basic heuristic require a large number of time-consuming FFT executions, this approach significantly slows down each ant's production of a completed resource allocation, which, in turn, limits the number of high-level iterations that can be performed within the CSC.

The success of combining a simple local search with GA and SA suggest that a more exhaustive local search may be worth investigating in future work. The more exhaustive local search might consider swapping applications between compute nodes in addition to moving applications between compute nodes. Although the introduction of swapping will increase the number of evaluations required to complete the local search procedure, it may lead to an improved result over the current coarse approach to local search.

6.6 Related Work

Often, modern parallel and distributed computing systems must operate in an environment replete with uncertainty while providing a required level of QoS. This reality inspired an increasing interest in robust resource allocation design in such systems. The following are some examples. The Robust Network Infrastructures Group at the Computer Science and Artificial Intelligence Laboratory at MIT takes the position that "... a key challenge is to ensure that the network can be robust in the face of failures, time-varying load, and various errors." The research at the User-Centered Robust Mobile Computing Project at Stanford "concerns the hardening of the network and software infrastructure to make it highly robust." The Workshop on Large-Scale Engineering Networks: Robustness, Verifiability, and Convergence (2002) concluded that the "Issues are ... being able to quantify and design for robustness ..." There are many other projects of similar nature at other schools and organizations.

In HC systems the concept of robust resource allocation called for a foundation of a universal robustness framework. The latter issue was first addressed in [3], where the authors proposed a four-step FePIA procedure to derive a robustness metric for a given resource allocation in a distributed system. The framework was based on *deterministic* estimates (e.g., current or nominal values) for each of the considered system parameters. As a measure of robustness, the "minimum robustness radius" was used that indicates the distance from the current (or nominal) state of the system in a multidimensional space of perturbation parameters to the nearest point where a QoS violation occurs. Assuming no a priori information available about the relative likelihood or magnitude of change for each perturbation parameter, the minimum robustness radius implies a deterministic worst-case analysis. In our stochastic model, more information regarding the variation in the perturbation parameters is assumed known. Representing the uncertainty parameters of the system as stochastic variables enables the robustness metric in the stochastic model to account for all possible outcomes for the performance of the system. An example comparison analysis between the stochastic robustness metric and deterministic one is given in [86]. The stochastic robustness metric requires more information and, in general, is far more complex

to calculate than its deterministic counterpart.

In [25], the robustness of a resource allocation is defined in terms of the schedule's ability to tolerate an increase in application execution time without increasing the total execution time of the resource allocation. A resource allocation's robustness implies system slack thereby the authors are focusing their metric on a single very important uncertainty parameter, i.e., variations in application execution times. Our metric is more generally applicable, allowing for any definition of QoS and able to incorporate any identified uncertainty parameters.

Our methodology relies heavily on an ability to model the uncertainty parameters as stochastic variables. Several previous efforts have established a variety of techniques for modeling the stochastic behavior of application execution times [18, 35, 64]. In [18], three methods for obtaining probability distributions for task execution times are presented. The authors also present a means for combining stochastic task representations to determine task completion time distributions. Our work leverages this method of combining independent task execution time distributions and extends it by defining a means for measuring the robustness of a resource allocation against an expressed set of QoS constraints.

In [51], a procedure for predicting task execution times is presented. The authors introduce a methodology for defining data driven estimates in a heterogeneous computing environment based on nonparametric inference. The proposed method is applied to the problem of generating an application execution time prediction given a set of observations of that application's past execution times on different compute nodes. The model defines an application execution time random variable as the combination of two elements. The first element corresponds to a vector of known factors that have an impact on the execution time of the application and is considered to be a mean of the execution time random variable. A second element accounts for all unmodeled factors that may impact the execution time of an application and is used to compute a sample variance. Potentially, this method can be extended to determine probability density functions describing the input random variables in our framework.

The deterministic robustness metric established for distributed systems in [3] was used

in multiple heuristics approaches presented in [86]. Two variations of robust mapping of independent tasks to machines were studied in that research. In the fixed machine suite variation, six static heuristics were presented that maximize the robustness of a mapping against aggregate errors in the execution time estimates. The variety of iterative algorithms such as Sum Iterative Maximization, Greedy Iterative Maximization, Genitor, and Memetic Algorithm, demonstrate higher performance as compared to the non-iterative greedy heuristics. However in deterministic domain, greedy heuristics required significantly less time to complete a mapping. A similar trade-off was observed for another variation where a set of machines must be selected under a given dollar cost constraint that will maximize the robustness of a mapping. In contrast, our previous study [87] demonstrated that greedy heuristics applied in a stochastic domain experienced a significant execution slowdown due to a substantial number of PMR calls required for evaluation at each step of a mapping "construction" process.

In [36], the authors present a derivation of the makespan problem that relies on a stochastic representation of task execution times. The paper demonstrates that the proposed stochastic approach to scheduling can significantly reduce the actual simulated system makespan as compared to some well known scheduling heuristics that are founded in a deterministic approach to modeling task execution times. The heuristics presented in that study were developed to minimize the expected system makespan given a stochastic model of task execution times. In contrast to [36] in our research, the emphasis was to build resource allocations capable of maintaining a certain level of probability to deliver on expressed QoS constraints by applying iterative resource allocation algorithms.

6.7 Conclusion

This chapter presented a set of iterative algorithms for finding stochastically robust resource allocations in heterogeneous clusters where workload surges are likely to occur. The objective function used in the design of our resource allocation techniques was based on our stochastic robustness metric. The resulting objective function was demonstrated to be suitable for evaluating the likelihood that a resource allocation will perform acceptably

in this environment, i.e., satisfy imposed QoS constraints, despite uncertainty in system parameters.

Multiple parameters pertaining to each iterative algorithm were setup for the best performance in the simulated environment which replicated a heterogeneous cluster-based processing center of a radar system. The goal in the conducted experiments was to generate a resource allocation that allows for the minimum time period between sequential sensor outputs and guarantees a specified probability level that data processing is completed in time.

The performance analysis of multiple test trials revealed a great potential of the GA algorithm to efficiently manage resource allocation in a large class of heterogeneous clusters operating on periodical data sets. In addition to complex radar systems, the developed workload distribution mechanism can be applied in other practical implementations such as surveillance for homeland security, monitoring vital signs of medical patients, and automatic target recognition systems.

CHAPTER 7

MAXIMIZING STOCHASTIC ROBUSTNESS OF STATIC RESOURCE ALLOCATIONS IN A PERIODIC SENSOR DRIVEN CLUSTER

7.1 Introduction

In parallel and distributed computing, multiple compute nodes are collectively utilized to simultaneously process a set of applications to improve performance over that of a single processor [26]. Often, such computing systems are constructed from a heterogeneous mix of machines that may differ in their capabilities, e.g., available memory, number of floating point units, clock speed, and operating system. This paper investigates robust resource allocation for a large class of heterogeneous computing (HC) systems that operate on *periodically updated* sensor data sets. Sensors (e.g., radar systems, sonar) in this environment produce new data sets at a fixed period Λ (see Fig. 23). Often, the period at which the sensor produces new data sets is fixed by the sensor and cannot be extended to account for long-running applications. Because these sensors typically monitor the physical world, the characteristics of the data sets they provide may vary in a manner that impacts the execution times of the applications that must process them. Suppose that each input data set must be processed by a collection of N independent applications that can be executed in parallel on the available set of M heterogeneous compute nodes. The goal of resource allocation heuristics in this environment is to allocate the N tasks to the M compute nodes such that all of the applications finish in less than Λ time units.

The allocation of compute nodes to applications can be considered static, i.e., all of the applications that are to be executed are known in advance and are immediately available for execution upon the arrival of a new data set. Because this is an HC system, the execution

---

A preliminary version of the research in this chapter appeared in [96].

times for each of the N independent applications differs across the M compute nodes. Resource allocation in such computing environments has been shown in general to be an NP-hard problem (e.g., [32, 50]). Thus, designing techniques for resource management in such environments is an active area of research (e.g., [1, 25, 26, 36, 72]).

In this environment, a new data set arrives every Λ time units. Thus, the completion time of the last to finish application must be less than or equal to Λ to ensure that the system is ready to begin processing the next data set upon its arrival. However, unpredictable variance in the characteristics of the input data sets may result in a significant change in the execution times of the applications that must process the data. This variance may cause the makespan of the resource allocation to exceed Λ, which is unacceptable. *Robust* design for such systems involves determining a resource allocation that can account for uncertainty in application execution times in a way that enables a probabilistic guarantee that all of the applications will complete within Λ time units.

The notion of stochastic robustness was established in [89] based on a mathematical model of HC systems. That research presented three greedy heuristics for resource allocation and three more sophisticated random search based algorithms. The emphasis of that research was on minimizing Λ subject to a constraint on the robustness of the resulting resource allocation.

Often, the required time period Λ is fixed by the sensor platform. In this work, we consider two cases for maximizing the robustness of resource allocations. In the first case,

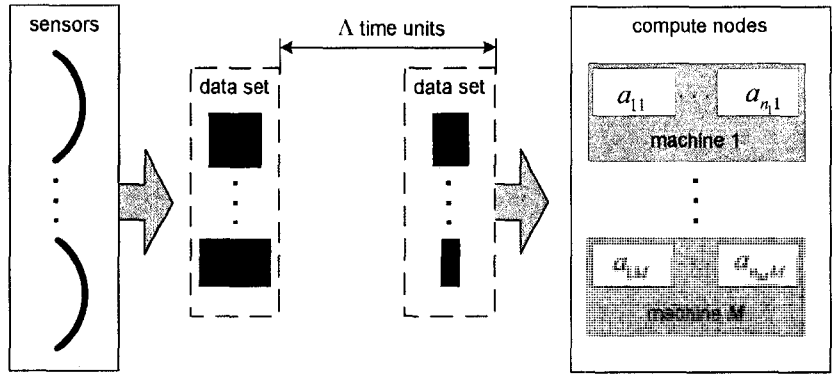


Figure 23: Major functional units and data flow for a class of systems that must periodically process data sets from a collection of sensors.

we consider a system where the Λ value is relatively loose. In this situation, we can apply simpler greedy heuristics to generate resource allocations with a non-zero probability of meeting the imposed deadline defined by Λ. In the second case, we consider tighter Λ values such that naïve greedy heuristics are incapable of producing resource allocations with a non-zero probability of meeting the deadline Λ. In this situation, the direct application of random search techniques, such as a genetic algorithm, are ineffective early in their search because naïve attempts to generate an initial population (typically based on random assignment or simple greedy techniques) also fail to produce resource allocations with a non-zero probability of meeting the provided deadline.

The major contributions of this paper include (1) the design of heuristic techniques that maximize the robustness of resource allocations subject to a constraint on the maximum allowable Λ, (2) a novel methodology for generating resource allocations with a non-zero probability of completing within a tight Λ time constraint, and (3) incorporating this methodology into heuristics.

The remainder of this work is organized in the following manner. Section 7.2 describes the model of stochastic compute node completion times used in this work. A brief introduction to the stochastic robustness framework is presented in Section 7.3 along with an introduction to using the stochastic robustness metric within a heuristic. Three random search based algorithms designed for this environment and one sample greedy heuristic are described in Section 7.4. The parameters of the simulation study used to evaluate a direct maximization of robustness are discussed in Section 7.5. Section 7.6 defines our methodology for satisfying timing constraints where simple greedy techniques produce allocations with no probability of success. This section presents the details of a steady-state genetic algorithm that successfully applies this approach for defining an initial population to maximize robustness through random search. A sampling of the relevant related work is presented in Section 7.7. Section 7.8 concludes the paper.

7.2 Stochastic Completion Times

In this environment, we are concerned with allocating a set of N independent applications to a collection of M dedicated heterogeneous compute nodes. Each of the N applications must process data that is produced by the system's sensors. In a static resource allocation, all of the applications to be executed are known in advance of performing a resource allocation. In this environment, although the data sets to be processed vary every Λ time units, the applications executed are the same, thus, the resource allocation environment can be considered static.

Application execution times are known to be data dependent. The data sets produced by the system sensors vary with changes in the real world, causing application execution times to vary in unpredictable ways. For this reason, we model the execution time of each application i ($1 \leq i \leq N$) on each compute node j ($1 \leq j \leq M$) as a random variable [104], denoted $\underline{\eta_{ij}}$. We assume that probability mass functions (pmfs) describing the possible execution times for each application on each compute node exist and are available for each η_{ij}. (See [64] for some techniques for estimating these pmfs.)

Using the application execution time pmfs, we can produce a completion time distribution for each compute node in a given resource allocation. The finishing time of each compute node is calculated as the sum of the execution time random variables for each application assigned to that compute node [89]. Let $\underline{n_j}$ be the number of applications assigned to compute node j. The pmf for the finishing time of compute node j, referred to as a local performance characteristic $\underline{\psi_j}$, can be expressed as follows:

$$\psi_j = \sum_{i=1}^{n_j} \eta_{ij}. \tag{46}$$

Thus, the system makespan, denoted $\underline{\psi}$, can be expressed in terms of the local performance characteristics as follows:

$$\psi = \max\{\psi_1, \cdots, \psi_M\}. \tag{47}$$

If the execution times η_{ij} for the applications assigned to compute node j are mutually independent, then the summation of Eq. 46 can be computed using an $(n_j - 1)$-fold convolution of the corresponding pmfs [63, 89].

7.3 Stochastic Robustness

The goal of each resource allocation heuristic is to assign applications to compute nodes such that the robustness of the resulting resource allocation is maximized subject to a constraint on the overall makespan (i.e., the completion time of the last to finish compute node). A robustness metric for this environment was derived using the FePIA procedure [3], in [89] and is summarized below.

For this system, the performance feature of interest is makespan, ψ. A resource allocation can be considered *robust* if the actual finishing time of each compute node is less than or equal to the periodicity of the data set arrivals Λ, i.e., $\psi \leq \Lambda$. We refer to this as the robustness requirement of the system.

Uncertainty in the system arises because the exact execution time for each application is not known in advance of its execution. That is, each of the execution time random variables η_{ij} is a source of uncertainty in a resource allocation. Because of its functional dependence on the execution time random variables, the system makespan is itself a random variable. That is, the uncertainty in application execution times can have a direct impact on the performance metric of the system.

To determine exactly how robust the system is under a specific resource allocation, we conduct an analysis of the impact of uncertainty in system parameters on our chosen performance metric. The stochastic robustness metric, denoted θ is defined as the probability that the performance characteristic of the system is less than or equal to Λ, i.e., $\theta = \mathbb{P}[\psi \leq \Lambda]$. For a given resource allocation, the stochastic robustness metric measures the probability that the generated system performance will satisfy our robustness requirement. Clearly, unity is the most desirable stochastic robustness metric value, i.e., there is a zero probability that the system will violate the established robustness requirement.

For each compute node j, the robustness of the local performance characteristic is found

as the sum over the impulses in the pmf describing ψ_j whose impulse values are less than or equal to Λ. Because there are no inter-task data transfers among the applications to be assigned, the random variables for the local performance characteristics $(\psi_1, \psi_2, \cdots, \psi_M)$ are mutually independent. As such, the stochastic robustness metric for a resource allocation can be found as the product of the probability that each local performance characteristic is less than or equal to Λ. Mathematically, this is given as:

$$\theta = \prod_{\forall j} \Big(\mathbb{P}[\psi_j \leq \Lambda] \Big). \tag{48}$$

Intuitively, the stochastic robustness of a resource allocation defines the probability that all of the applications will complete within the allotted time period Λ. In this research, we are provided a constraint on the maximum time period required to process all applications, i.e., Λ, and attempt to derive a resource allocation to maximize the probability that all of the applications will complete by this deadline.

7.4 Heuristics

7.4.1 Two-Phase Greedy

The Two-Phase Greedy heuristic is based on the principles of the Min-Min algorithm (first presented in [50], and shown to perform well in many environments, e.g., [26, 73]). The heuristic requires N iterations to complete, resolving a single application to compute node assignment during each iteration.

In the first phase of each iteration, the heuristic determines the application to compute node assignment that minimizes the expected completion time for each of the applications left unmapped. In the second phase, the heuristic selects the application to compute node assignment from the first phase that has the smallest expected completion time. Pseudo-code for the Two-Phase Greedy heuristic is provided in Figure 24.

In this heuristic, we chose to minimize the expected completion time of applications as opposed to maximizing robustness because early in an allocation, many of the possible application/compute-node assignments will have identical robustness values, i.e., unity.

```
while not all applications are mapped
    for each unmapped application a_i find the compute node m_k such that
        k ← argmin[1]_{1≤j≤M} {E[ψ_j(a_i)]};
        resolve ties arbitrarily;
    from all (a_i, m_k) pairs found above select pair (a_x, m_y) such that
    (a_x, m_y) ← argmin_{(a_i,m_k)} {E[ψ_k(a_i)]};
    resolve ties arbitrarily;
    map application a_x to compute node m_y;
end of while
```

Figure 24: Pseudocode for the Two-Phase Greedy heuristic. We use the shorthand notation, $\psi_j(a_k)$, to denote an evaluation of the local performance characteristic ψ_j given the set of applications already assigned to compute node j with the addition of application a_k.

This happens because many of the applications have yet to be assigned, therefore, the subset that have already been assigned are all guaranteed to finish before Λ (i.e., SRM = 1). Thus, an application of the Min-Min algorithm that directly maximizes robustness in this environment in ineffective.

7.4.2 Genetic Algorithm

The adopted genetic algorithm (GA) was motivated by the Genitor evolutionary heuristic first introduced in [105]. Our GA implementation models a complete resource allocation as a sequence of numbers, referred to as a chromosome, where the i^{th} entry in the sequence corresponds to the compute node assignment for the i^{th} application, denoted $\underline{a_i}$. The ordering of applications in a chromosome is not significant for this environment and can be considered arbitrary. The fitness of each chromosome is determined according to the robustness of the resource allocation it represents. That is, a higher robustness value θ indicates a more fit chromosome.

In our implementation, we use a sorted list of the 100 most fit chromosomes encountered, referred to as the population, where the chromosomes are sorted according to robustness. The appropriate size of the population was selected through experimentation.

[1] Argmin is an operation that returns the value of the argument for which the given expression attains its minimum value.

The initial members of the population were generated by applying the simple greedy heuristic from [89] and applying a local search operator to the result (the details of the local search operator are defined below). The simple greedy heuristic randomly selects an application and assigns it to the compute node that provides the smallest completion time. After a chromosome is produced by this simple greedy heuristic, we apply the local search operator to produce a new chromosome that is a local minimum, i.e., the robustness of the chromosome cannot be further improved upon using our local search procedure. Before the generated chromosome can be inserted into the population, a check is performed to ensure that the new chromosome is unique. In this way, we ensure that no member of the initial population is overrepresented. This chromosome generation process is repeated until the population size reaches its limit, i.e., 100 chromosomes.

Our GA operates in a *steady state* manner, i.e., for each iteration of the GA only a single pair of chromosomes is selected from the population for crossover [81]. Chromosome selection is performed using a linear bias function [105], where the rank of each chromosome is determined by its fitness. Each new chromosome generated is inserted into the population in sorted order according to its fitness value. For each chromosome inserted, the least fit chromosome in the population is removed, so that the size of the population is held fixed. To maintain a diverse population, the insertion process prevents duplicate chromosomes from being inserted into the population.

The crossover operator was implemented using the two-point reduced surrogate procedure [105], where the parent chromosomes are compared to identify the entries that differ between them. Crossover points are randomly selected such that at least one element of the parent chromosomes differs between the selected crossover points to guarantee offspring that are not clones of their parents. Each execution of the crossover operator will produce two new offspring.

Following crossover, a local search procedure, conceptually analogous to the steepest descent technique [30], is applied to each of the produced offspring prior to their insertion into the population. The local search implemented in our GA is similar to the coarse refinement presented as part of the GIM heuristic in [100]. Local search relies on a simple

four-step procedure to maximize θ relative to a fixed Λ value. First, for a given resource allocation, the compute node with the lowest individual probability to meet Λ is identified. From the applications assigned to this compute node, local search identifies the application that, if moved to a different machine, would increase θ the most. This requires re-evaluating θ every time an application-compute node re-assignment is considered. Once an application-compute node pair has been identified, the chosen application is moved to its chosen compute node. If no application can be moved to a new compute node to attain an improvement in θ, then we consider swapping two applications. One of the two applications considered for swapping must currently be assigned to the compute node under consideration, selecting the alternate application that improves θ the most.

The procedure repeats from the first step until there are no application moves from the lowest probability compute node that would improve θ. The procedure is then repeated for each compute node in increasing order of the probability that the local performance characteristic is less than or equal to Λ until no application can be moved to improve θ. For this procedure, it is assumed that $\theta < 1$; otherwise, no improvements can be made through local search.

The final step within each iteration of our GA implementation applies a mutation operator. For each iteration of the GA, the mutation operator is applied to a small percentage of the overall population, referred to as the mutation selection rate. In our implementation, we chose a mutation selection rate of 0.1, i.e., during each iteration we selected 10% of the population for mutation. We arrived at this mutation selection rate through experimentation, i.e., given the details of our simulations a selection rate of 10% produced the best results. During mutation, each application assignment within the chromosome to be mutated is individually modified with a probability referred to as the mutation rate. For the simulated environment, the best results were achieved using a mutation rate of 0.02, determined through experimentation. Once an application-compute node assignment has been selected for mutation, the mutation operator randomly selects a different compute node assignment for the chosen application. After mutation, the local search operator is applied to the new chromosome and the result is inserted into the population.

```
generate initial population using simple greedy heuristic and local search;
rank population based on θ values;
while eval count < 4,000,000
   select two chromosomes from the population for crossover;
   select crossover points;
   exchange compute node assignments between crossover points;
   apply local search to the two offspring;
   insert resulting offspring into population
    if not already present in the population;
   if needed, remove worst chromosome(s)
    to maintain fixed population size;
   for each chromosome in the population
     generate a random number x in the range [0,1];
     if x < mutation selection rate
       for each application in the selected chromosome;
         generate a random number y in the range [0,1];
         if y < mutation rate
           arbitrarily change compute node assignment
           of the selected application;
       apply local search to resulting chromosome;
       insert resulting offspring into population
        if not already present in the population;
       if needed, remove worst chromosome(s)
        to maintain fixed population size;
end of while
output the best solution encountered.
```

Figure 25: Pseudo-code for the GA heuristic.

We limited the total number of chromosome evaluation function calls in our GA implementation to 4,000,000, because for the simulation trials tested, this number of chromosome evaluation function calls enabled the GA to find a resource allocation that provided a near unity robustness value. The GA procedure is summarized in Fig. 25.

7.4.3 Genetic Algorithm with Path Relinking (GAPR)

Within the context of a genetic algorithm, path relinking can be used as a means of combining two "parents" to form a new type of crossover operator [81]. The path relinking implementation of crossover begins with a pair of chromosomes, and draws a path from one parent to the other by exchanging one application/compute-node assignment at a time.

Our implementation of path relinking begins with a pair of chromosomes to be used as parents in the operation. One is labeled the *start* chromosome and the other is labeled the *end* chromosome. Next, we initialize the "*best* chromosome encountered" to the parent chromosome with the highest robustness value and initialize the "*current* chromosome" to the start chromosome. For each application i ($i = 1, \cdots, N$), in the current chromosome, we change the compute node assignment of a_i in the current chromosome to the assignment of a_i in the end chromosome. Next, we copy the current chromosome into a temporary location and apply local search to this temporary chromosome. If the compute node assignments for a_i are the same in each parent, then we skip the local search procedure because it will produce the same result as the previous application of local search. Finally, if the robustness of the chromosome produced by the local search is greater than that of the best chromosome encountered, then we replace best with this result. We then iterate i and the procedure continues in this manner until the current chromosome is identical to the end chromosome. When this occurs, the procedure outputs the best chromosome as the offspring of the operation and exits. To generate a second offspring from this operator, we proceed as before, but we exchange the start and end chromosomes to follow a second path.

An example of the path relinking procedure is depicted in Figure 26. In the figure, two initial chromosomes y and x are connected by transforming chromosome y into chromosome x. For each intermediate point along the path from y to x, we apply the local search procedure to find the resource allocation whose robustness defines a local optimum. Because all of the members of our steady-state GA population are local optima, the chromosomes y and x are themselves local optima. By transforming solution y into solution x one compute node assignment at a time, path relinking encounters an intermediate chromosome that is in the basin of attraction defined by the local optimum o. Thus, the local search starting

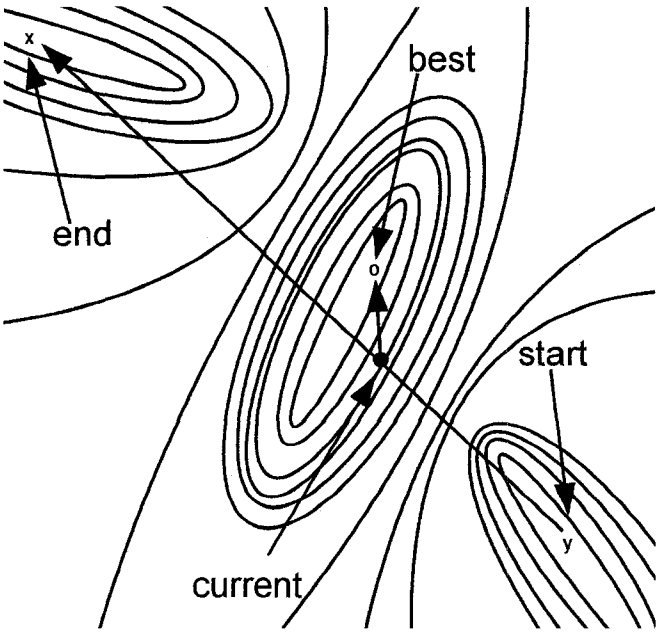


Figure 26: A conceptual depiction of the path relinking procedure. Each contour line in this depiction corresponds to the same robustness value. The search begins at chromosome y and generates a path to chromosome x using the path relinking procedure. Local search is applied to each intermediate chromosome produced along the path from y to x. The path connecting y and x passes into a basin of attraction leading to an improved solution o.

from this intermediate chromosome produces o as an output. Further, if the fitness of o is greater than that of both y and x, then o may be one of the offspring produced by this path relinking crossover operation. In our implementation, we produce only a single offspring from each direction of the path relinking operator.

We form a new GA heuristic based on the steady-state GA presented in the previous subsection by replacing the two-point reduced surrogate crossover operator with path relinking. Pseudo-code for our implementation of path relinking is provided in Figure 27.

7.4.4 Simulated Annealing

The Simulated Annealing (SA) algorithm—also known in the literature as Monte Carlo annealing or probabilistic hill-climbing [73]—is based on an analogy taken from thermodynamics. In SA, a randomly generated solution, structured as the chromosome from our GA, is iteratively modified and refined. SA in general can be considered a random search technique that operates with one possible solution (resource allocation) at a time.

```
start ← first chromosome selected for crossover;
end ← second chromosome selected for crossover;
if θ(start) > θ(end)
  best ← start;
else
  best ← end;
current ← start;
for i = 1 to N
  if a_i in start ≠ a_i in end
    a_i in current ← a_i in end;
    result ← apply local search to current;
    if θ(result) > θ(best)
      best ← result;
end of for
offspring ← best;
output offspring;
```

Figure 27: Pseudo-code of one direction of the path-relinking crossover procedure.

To deviate from the current solution in an attempt to find a better one, SA repeatedly applies the mutation operation of our steady-state GA, applying the local search operator to the mutated result prior to its evaluation. The mutation rate was set to 10% for our simulation trials. Once a new solution, denoted S_{new}, is produced, a decision regarding the replacement of the previous solution, denoted S_{old}, with S_{new} has to be made. If the fitness of the new solution, denoted $\theta(S_{new})$, found after evaluation, is higher than the old solution, the new solution replaces the old one. Otherwise, SA will probabilistically allow poorer solutions to be accepted during the search process, which makes this algorithm different from other strict hill-climbing algorithms [73]. The probability of replacement is based on a system temperature, denoted $\mathbb{T}$, that decreases with each iteration. As the system temperature "cools down," it becomes more difficult for poorer solutions to be accepted. Specifically, the SA algorithm selects a random number from the range $[0, 1)$ according to a uniform distribution. If

$$random[0,1) > \frac{1}{1+\exp\left(\frac{\theta(S_{new})-\theta(S_{old})}{\mathbb{T}}\right)}, \tag{49}$$

then the new poorer resource allocation is accepted; otherwise, the old solution is kept. As can easily be seen in Eq. 49, the probability for a new solution of similar quality to be

accepted is close to 50%. In contrast, the probability that a poorer solution is rejected is rather high, especially when the system temperature becomes relatively small.

After each mutation (described in Subsection 4.3) and subsequent local search, the system temperature $\mathbb{T}$ is reduced to 99% of its current value. This percentage, defined as the cooling rate, was determined experimentally by varying the cooling rate in the range $[0.9, 1)$. The fitness of each chromosome, θ, is inherently bound to the interval $[0, 1]$. Consequently, only small differences between $\theta(S_{new})$ and $\theta(S_{old})$ are possible, causing Eq. 49 to remain very near 0.5 for large values of $\mathbb{T}$. Based on our experimentation, we set the initial system temperature in Eq. 49 was to 0.1.

For the simulation trials tested, our implementation of SA was terminated after 4,000,000 chromosome evaluation function calls. Limiting the overall number of chromosome evaluation function calls to 4,000,000 enabled a fair comparison with the results of both of our steady-state GA implementations. The SA procedure is summarized in Fig. 28.

```
S_old ← initial Two-phase greedy solution;
𝕋 ← 0.1;
while eval count < 4,000,000
    S_new ←mutate S_old;
    apply local search to S_new;
    if θ(S_new) > θ(S_old)
        S_old ← S_new;
    else if Eq. 49 holds
            S_old ← S_new;
𝕋 ← 0.99 × 𝕋;
end of while
output S_old;
```

Figure 28: Pseudo-code for the Similated Annealing heuristic.

7.5 Simulation Study

7.5.1 Setup

For our simulations, the periodicity Λ of data set arrivals was assumed fixed at 540 time units. The value for the constraint Λ was selected to present a challenging resource allocation problem for our chosen heuristics (i.e., the resulting θ was neither 1 nor 0) based on the number of applications, the number of compute nodes, and the execution time pmfs used for η_{ij}. The goal of the heuristics is to find resource allocations that have the highest probability to complete all of the applications within the given time period Λ.

To evaluate the performance of the heuristics described in Section 7.4, the following approach was used to simulate a cluster-based system for processing radar data sets. The execution time distributions for 28 different types of possible radar ray processing algorithms on eight ($M = 8$) heterogeneous compute nodes were generated by combining experimental data with benchmark results. The experimental data, represented by two execution time sample pmfs, were obtained from experiments conducted on the Colorado MA1 radar [53]. These sample pmfs contain application execution times for 500 different radar data sets of varying complexity by the Pulse-Pair & Attenuation Correction algorithm [22] and by the Random Phase & Attenuation Correction algorithm [22]. Both applications were executed in non-multitasking mode on the Sun Microsystems Sun Fire V20z workstation. To simulate the execution of these applications on a heterogeneous computing system, each sample pmf was scaled by a performance factor corresponding to the performance ratio of a Sun Microsystems Sun Fire V20z to each of eight selected compute nodes[1] based on the results of the fourteen floating point benchmarks from the CFP2000 suite [98]. Combining the results available from the CFP2000 benchmarks with the sample pmfs produced by the two available applications provided a means of generating the 28×8 matrix of application execution times, where the kj^{th} element in the matrix corresponds to the application execution time pmf of a possible ray processing algorithm of type k on compute node j.

[1]The eight compute nodes selected to be modeled were: Altos R510, Dell PowerEdge 7150, Dell PowerEdge 2800, Fujitsu PRIMEPOWER 650, HP Workstation i2000, HP ProLiant ML370 G4, Sun Fire V65x, and Sun Fire X4100.

Each simulation trial consisted of a set of 128 applications ($N = 128$) to be assigned to any one of the eight available heterogeneous compute nodes. To evaluate the performance results of each heuristic, 50 simulation trials were conducted. For each *trial*, the type of each application was determined by randomly sampling integers in the range $[1, 28]$.

7.5.2 Results

The results of the simulation trials are presented in Figure 29. The 50 simulation trials provide a good estimate of the mean as demonstrated by the 95% confidence intervals for the results of the resource allocation techniques. The GA, GAPR, and SA heuristics were able to improve upon the robustness values achieved by the Two-Phase Greedy solution. For the 50 conducted trials, the Two-Phase Greedy heuristic produced an average robustness value of 67.63%. The SA results showed a mean robustness value of 94.02%. Including the path relinking crossover operator in the GA, we were able to produce an average robustness value of 96.62%. The regular steady-state GA performed better on average than any of the other heuristics with an average robustness value of 98.15%. The confidence intervals for the GA, GAPR, and SA heuristic results overlap. Given their values, their performance may be similar because they are all near the achievable optimal value. The comparable performance of the SA and the two GA variants may suggest that the local search operator, common to all of the techniques, may have a significant impact on the results attained.

There was a significant difference between the heuristics in terms of their worst-case completion time performance. The worst-case completion time performance for a resource allocation corresponds to the largest possible completion time from the completion time distributions taken across all compute nodes in the allocation. This value is significant because it also corresponds to a robustness value of 1, i.e., the smallest value of Λ for which the resource allocation is guaranteed to complete. For our simulation study, the regular steady state GA had a mean worst case completion time of 543.25 time units with a 95% confidence interval of plus or minus 6.09 time units. The mean worst case completion time for the GA with path relinking was found to be 550.57 time units with a 95% confidence interval of plus or minus 5.6 time units. The SA mean worst case completion time was

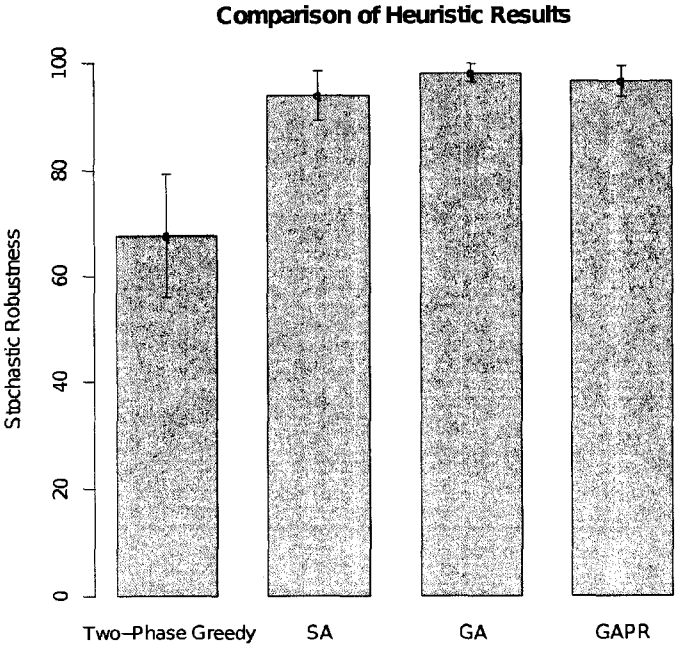


Figure 29: A comparison of the average robustness values over 50 trials attained through simulation for the GA with Path Relinking (GAPR), GA, SA, and Two-Phase Greedy heuristics.

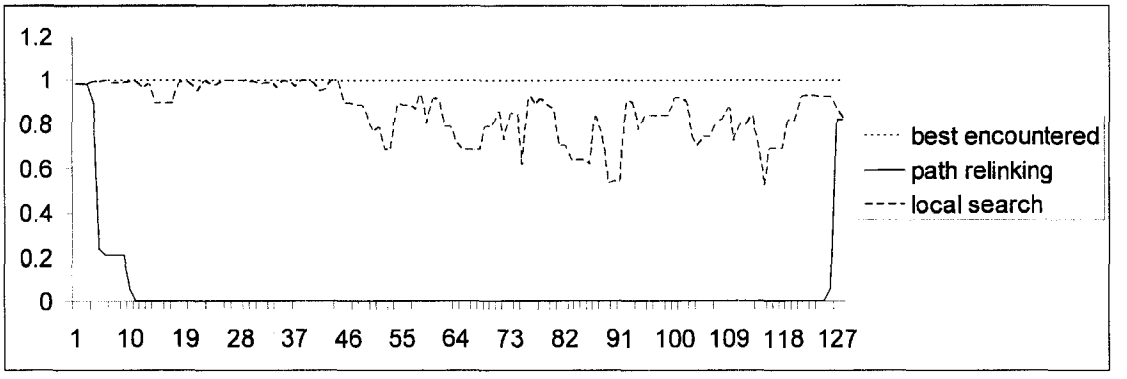


Figure 30: An example result obtained by applying the path relinking crossover procedure in one direction.

found to be 564.84 time units with a 95% confidence interval of plus or minus 4.27 time units. Thus, comparing the three approaches on the basis of mean worst case completion time shows a significant difference between the SA heuristic and the two variants of the steady-state GA.

To better visualize the path-relinking procedure, we plotted the robustness attained by each step of the operation. Figure 30 shows the results of an example application of the path relinking crossover operator. The result was obtained by transforming one chromosome into another using our defined path relinking procedure. After each application to compute node assignment is modified, the local search operator is applied to the intermediate solution. The dashed line of the figure corresponds to the robustness of the current chromosome before local search has been applied and the solid line corresponds to the robustness of the allocation after applying local search. The dotted line provides an indication of the highest robustness value encountered during this run of path relinking. The first robustness value of the dashed line corresponds to the first parent in the crossover and the last robustness value corresponds to the robustness value of the alternate parent. In this example, the initial allocation of the first parent had a robustness value of 0.9808 and the alternate parent had a robustness of 0.8224. Applying the path relinking crossover operator produced a final allocation with a robustness value of 0.9981.

7.6 *Maximizing Robustness for Tighter Λ Values*

7.6.1 Overview

Maximizing robustness using the random-search based approaches described earlier can function properly only if the robustness of the initial and intermediate solutions are non-zero. That is, in all of the presented random search techniques, intermediate resource allocations are directly compared to one another based on robustness and ranked accordingly. Thus, if the robustness of two allocations are equal to zero, then we cannot discriminate between them based on robustness alone. For this case, we would like to identify an alternative metric for evaluating resource allocations that can always discriminate between resource allocations whose robustness value is zero.

The metric that we use is the expected makespan of a given resource allocation, which allows us to differentiate between solutions that both have a zero robustness value. Using a GA to find chromosomes that will minimize the expected makespan may lead us to solutions with non-zero robustness values.

In the next sub-section, we prove that if the expected makespan for a given resource allocation is less than or equal to Λ, then the stochastic robustness value of that allocation will be non-zero. In Subsection 7.6.3, we define a resource allocation technique based on a steady-state genetic algorithm that minimizes the expected makespan of resource allocations to produce an initial population for our "robustness" based GA. This section concludes with the results in Subsection 7.6.4.

7.6.2 Alternative Metric for Evaluating Allocations

Define the *expected* makespan of a resource allocation, denoted $\underline{\mu}$, as the makespan of the resource allocation as found using the expectation of machine completion time distributions. Let $\underline{\mu_j}$ be the expected completion time of machine j, i.e., $\mathbb{E}[\psi_j] = \mu_j$, then μ can be found by taking the max over all expected completion times,

$$\mu = \max_{1 \leq j \leq M} \mu_j. \tag{50}$$

A useful property of the expected makespan of an allocation is that if $\mu \leq \Lambda$ then θ is bounded below by $(0.5)^M$, i.e., the robustness of the allocation is guaranteed to be non-zero. To prove this property, assume that $\mu \leq \Lambda$. Recall that μ is the maximum expected completion time over all machines, which implies that $\mu_j \leq \mu \leq \Lambda$ $(\forall j)$. Define $\underline{\theta_j}$ to be the robustness of the completion time distribution for machine j relative to Λ. From our assumption that $\mu \leq \Lambda$ and the definition of μ, we know that for some machine k the largest that μ_k can be is Λ, which corresponds to the smallest value of θ_k. If $\mu_k = \Lambda$, then θ_k must equal 0.5 because μ_k is the mean of the completion time distribution. From our definition of μ, for each machine j, $\mu_j \leq \mu_k$ implying $\theta_j \geq 0.5$ $(\forall j)$. Thus,

$$\theta = \prod_{1 \leq j \leq M} \theta_j \geq (0.5)^M. \tag{51}$$

Therefore, if there are M compute nodes with $\mu \leq \Lambda$, then $\theta \geq (0.5)^M$. This proves that if the expected makespan of a resource allocation is less than or equal to Λ, then the robustness of that allocation is non-zero. For cases based on pmfs, because the expected value may not be a possible outcome, we replace the expected value of the pmf with the smallest possible outcome that is greater than the expected value, ensuring that Equation 51 still holds.

If we can establish an initial population where each resource allocation satisfies $\mu \leq \Lambda$, then our earlier robustness GA heuristics can be applied to cases where Λ is tight. We make the simplifying assumption that Λ has been selected such that resource allocations exist that can satisfy this constraint on μ.

To establish an initial population using this method, we can employ simpler minimization techniques that operate on the mean execution times of applications as opposed to the entire distribution of possible execution times. In the next subsection, we describe a GA that minimizes the expected makespan of each resource allocation (referred to as the makespan GA).

7.6.3 GA for Establishing Initial Population

In our original robustness GA implementation, we employed a simple greedy heuristic to generate the initial population. Unfortunately, if the Λ constraint is too small, then this simple method of population generation will not produce solutions with non-zero robustness values. Based on the discussion of the previous subsection, we implemented a makespan GA that minimizes the mean makespan of allocations. The output of the makespan GA is an initial population whose members all have a non-zero robustness value, and so can be used as the initial population in the robustness GA.

The implementation of the makespan GA operates in a manner similar to the robustness GA, using the path relinking crossover operator and local search (but based on minimizing the expected makespan). The fitness of each resource allocation in the makespan GA is measured by the expected makespan of the allocation instead of its robustness. Based on our earlier analysis, if the expected makespan can be reduced to at least Λ, then the resulting robustness of the allocation will be non-zero. The makespan GA terminates when the expected makespan of the worst member of the population is less than or equal to Λ or a maximum number of iterations has elapsed. If the makespan GA terminates because a maximum number of iterations has elapsed, then the robustness GA can be populated with any members whose $\mu \leq \Lambda$. In addition, it may be possible to identify chromosomes where $\mu > \Lambda$ and the robustness of the solution is non-zero. In our simulations, the makespan GA was always able to find a complete initial population whose expected makespan was less than or equal to Λ.

After the makespan GA terminates, we are left with a population whose robustness values are all non-zero. However, the order of the chromosomes is based on the expected makespan, as opposed to robustness. Thus, before the population can be used by our robustness GA heuristic, we must evaluate each chromosome based on robustness and re-sort the population. Although the makespan GA applied local search to each member of the population, the local search was based on the expected makespan. To ensure that the population consists only of local minima relative to robustness, we apply the robustness based local search operator to each chromosome, then insert the resulting chromosome into

the population. It is possible that after applying robustness based local search to each chromosome and rebuilding the population that some of the chromosomes will result in clones (i.e., duplicate chromosomes). Because the insertion procedure of the robustness GA explicitly disallows clones, the initial population in our robustness GA may begin with fewer chromosomes than our desired population size. However, the population will typically steadily grow back to the predetermined population size in the first few rounds of the robustness GA and will remain fixed at our chosen population size for the remainder of the simulation.

7.6.4 Results for Tighter Λ Values

To evaluate the effectiveness of generating an initial population for tighter Λ values using the makespan GA approach, we re-ran the simulation trials with a new Λ value that is tailored to each simulation trial. In these trials, the Λ constraint was uniquely set for each simulation trial based on the collection of tasks to be executed for that trial. In [89], we presented a mechanism for calculating a lower bound on Λ. In this work, we applied the lower bound calculation to each trial and used 120% of a lower bound value as the Λ constraint. This resulted in an average Λ constraint of 506 time units as opposed to the fixed Λ constraint of Section 7.5.2, i.e., $\Lambda = 540$.

For this tighter Λ constraint, the direct robustness maximization techniques in Section 7.4 were unable to find solutions with non-zero robustness values. Using the same number of chromosome evaluations, this modified approach that used both the makespan GA and the robustness GA, denoted GAPR2, was able to generate solutions with an average robustness value of 89.6% with a 95% confidence interval of plus or minus 2 percentage points. These results clearly indicate the success of the proposed approach for generating resource allocations with high robustness values under tighter Λ constraints.

As a final comparison, we compared the results of the GAPR2 heuristic to those of our direct methods for simulations where Λ was set to 540, as was done in Section 7.5.2. For these simulations, GAPR2 performed comparably to GAPR, suggesting that the GAPR2 heuristic is capable of performing well for both loose Λ constraints and tight Λ constraints.

7.7 Related Work

A universal framework for defining the robustness of resource allocations in heterogeneous computing systems was addressed in [3]. This work referred to the ability of a resource allocation to tolerate uncertainty as the *robustness* of that resource allocation and established the FePIA procedure for deriving a deterministic robustness metric. In [89], the authors used the FePIA procedure to define a robustness metric for static stochastic resource allocation environments. The research in [89] focused on minimizing the makespan of a stochastic resource allocation subject to a constraint on the robustness of that allocation. In this current paper, we have shown that it is possible to instead directly maximize the robustness of a resource allocation given a constraint on the allowed makespan.

In [25], the problem of robust resource allocation was addressed for scheduling directed acyclic graphs (DAGs) in a heterogeneous computing environment. Robustness was quantitatively measured as the "critical" (i.e., the smallest) slack among all components that comprise a given DAG. The authors focused on designing resource allocations that maximized robustness for a deterministic environment. Our robustness metric is based on stochastic information about the uncertainties.

Our methodology requires that the uncertainty in system parameters can be modeled as stochastic variables. A number of methodologies exist for modeling the stochastic behavior of application execution times (e.g., [18,35,64]). In [18], a method is presented for combining stochastic task execution times to determine task completion time distributions. Our work leverages this method of combining independent task execution time distributions and extends it by defining a means for measuring the robustness of a resource allocation against an expressed set of QoS constraints.

In [36], the authors demonstrate the use of a GA to minimize the expected system makespan of a resource allocation in a heterogeneous computing environment where task execution times are modeled as random variables. This research demonstrates the efficacy of a stochastic approach to resource scheduling, by showing that it can significantly reduce system makespan as compared to some well known scheduling heuristics that are based on a deterministic modeling of task execution times. The heuristics presented in that study were

used in the stochastic domain to minimize the expected system makespan given a stochastic model of task execution times, i.e., the fitness metric in that approach was based on the first moment of random variables. The emphasis of our approach is on quantitatively comparing one resource allocation to another based on the stochastic robustness metric, i.e., the probability of satisfying a given makespan constraint. The success of the Genetic Algorithm applied to stochastic resource allocation in [36] was a motivating factor for our selection of a Genetic Algorithm in this study; however, our GA methodology differs significantly from that in [36], for both the robustness GA and the makespan GA.

7.8 Conclusions

This research presented three distinct heuristics for directly maximizing the robustness of a resource allocation. The GA, GAPR, and SA techniques were shown to significantly outperform a simpler Two-Phase greedy heuristic. A comparison of the three heuristics revealed the great potential for the GA and SA algorithms to efficiently manage resources in distributed heterogeneous systems operating under uncertainty.

We also proposed a new method for applying these heuristics when the stochastic robustness metric indicates that all of the initial resource allocations found by the greedy heuristic have zero probability of finishing in less than Λ time units. We applied this methodology to create the makespan GA heuristic for generating an initial population for use in our robustness GA technique that directly maximizes the robustness of resource allocations subject to a constraint on the makespan of the solution. Our simulations clearly indicate the viability of this combined approach to maximizing the stochastic robustness metric of resource allocations.

CHAPTER 8

MEASURING THE ROBUSTNESS OF RESOURCE ALLOCATIONS IN A STOCHASTIC DYNAMIC ENVIRONMENT

8.1 Introduction

Heterogeneous parallel and distributed computing is defined as the coordinated use of compute resources that have different capabilities to optimize system performance features. Often, heterogeneous, distributed computing systems must operate in an environment replete with uncertainty. Robustness in this context can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed [3]. We present the use of a stochastic robustness metric [86] to quantify the robustness of a resource allocation in a *dynamic* environment. This formulation of the stochastic robustness metric is used to predict the typical relative performance of three different resource allocation heuristics taken from the literature and adapted to the presented problem. A Bayesian regression model is fit to the combined results of the three heuristics to demonstrate the relationship between the stochastic robustness metric and the presented performance metric. The accuracy of the robustness predictions are then evaluated to determine their utility in predicting resource allocation heuristic performance in a dynamic environment.

The major contribution of this chapter is a mathematical formulation for quantifying

The research presented in this chapter was jointly conducted with my colleagues Louis Briceño, Timothy Renner, Vladimir Shestak, Joshua Ladd, Andrew Sutton, David Janovy, Sudha Govindasamy, Amin Alqudah, Rinku Dewri, and Puneet Prakash [93].

the robustness of a resource allocation in a stochastic dynamic environment and a methodology for applying the robustness formulation to predict heuristic performance. We demonstrate the proposed methodology by successfully predicting the relative performance of three heuristics taken from the literature and adapted to this environment. Finally, we present a detailed evaluation of the three heuristics using both the proposed robustness metric and a performance objective appropriate to the studied environment.

The environment considered in this research is that of a heterogeneous, distributed computing system designed to service a high volume web site of world-wide interest. The system being modeled was used to implement the 1998 World Cup web site [8] that processed more than 1.3 billion HTTP requests during the summer of 1998. The web site was provided to a world-wide audience by four heterogeneous, geographically dispersed systems—each with their own processing capacity and workload distribution techniques. This class of system is very challenging to implement but occurs surprisingly frequently. The World Cup football tournament is just one example of an event of world-wide appeal that necessitates web based coverage. Sites of this type are typically constructed for specific events and the volume of traffic is on the order of billions of requests in a period of only a few months or less. Other such events might include the Summer Olympics, the Winter Olympics, or the tour de France; all are reasonably expected to draw the attention of a world-wide audience in the billions.

This research developed resource allocation heuristics for a single location of a distributed system capable of processing a high volume of requests. Incoming requests were dispersed to one of four processing centers. Thus, each location was responsible for processing some fraction of the total traffic to the site. This work focused on a location comprising eight heterogeneous servers responsible for processing 45% of the overall traffic [7].

A task is defined to be a menu-driven HTTP request for data from the web site. Mapping tasks to machines in this distributed system is rather challenging as it must be done under uncertainty because the *exact* execution time required to process a task is not known *a priori*. However, past observations of task execution times can be used to construct a probability mass function (pmf) [104] that models the possible execution times for a given

task. The pattern of task arrivals to the site was modeled after real traffic patterns observed by the 1998 World Cup web site [7].

In the next section, we present the details of the problem to be addressed. Section 8.3 defines a means for determining stochastic completion times for tasks in this system using the details of the problem statement. The definition of stochastic completion times is then used in Section 8.4 to derive a stochastic robustness metric that is subsequently used to predict heuristic performance. In Section 8.5, we present the heuristics that have been developed as part of this research. Details of the simulation environment are presented in Section 8.6 and the results of the heuristics are evaluated in Section 8.7. Section 8.8 presents an overview of the relevant related work. Section 8.9 concludes the chapter.

8.2 Problem Statement

8.2.1 Introduction

The system studied in this research is an instance of a more general class of dynamic, heterogeneous computing (HC) system where task arrival times are not known in advance and exact task execution times are uncertain prior to their completion. All incoming tasks to the system are assumed to have been previously classified into one of C classes prior to their arrival. Each of the classes corresponds to a gross classification of the relative complexity of the request being processed. Each task class defines a set of pmfs, where each pmf describes the probability of all execution times for that class on a given machine within the HC suite. Further, all of the classes of tasks (HTTP requests) that the system may be asked to perform are known in advance, i.e., the web server has prior knowledge about what web pages it is providing.

Each arriving task has a relative deadline limiting the total time available to process each request. The relative deadline for each task class is assumed to have been established in advance. Because tasks in this environment are HTTP requests for data, made by a user of the website, it is assumed that if a task cannot be completed by its deadline then the request can be considered to be "timed out" and the user that submitted the original request will make the request again. Therefore, there is no benefit to completing tasks that

miss their deadlines and, consequently, tasks that miss their deadlines will be discarded.

8.2.2 Performance Metric

In this environment, each incoming task has a hard deadline for its completion, i.e., failure to complete a task by its deadline will result in a penalty. To model the impact of missing a task deadline the resource allocation heuristic will be penalized by a constant factor of 1 for each task deadline that is missed. That is, for a given task i with deadline β_i^{max} let $comp(i)$ be the actual completion time of task i and define the cost to process a task i, denoted $cost(i)$, as follows

$$cost(i) = \begin{cases} 0, & \text{if } comp(i) \leq \beta_i^{max}; \\ 1, & \text{otherwise.} \end{cases} \tag{52}$$

The overall cost of a resource allocation is defined as the sum of the cost of each processed task. Define PT as the set of all tasks that are processed by the system, then the objective of a resource allocation in this environment can be expressed as,

$$\text{Minimize} \sum_{\forall i \in PT} cost(i). \tag{53}$$

Intuitively, resource allocations in this environment are expected to minimize the number of tasks that miss their deadlines.

8.2.3 Mapping Events

All of the resource allocation heuristics evaluated in this work operate in a pseudo-batch mode [66, 72]. In a pseudo-batch mode heuristic, all tasks that have not begun execution and are not next in line to begin execution can be considered for remapping when a mapping event occurs. Mapping events occur within the system whenever a new task arrives or an existing task completes.

8.3 Stochastic Task Completion Time

Although some tasks may belong to the same class, task execution times may still vary depending on the details of the data requested. For this reason, task execution times are modeled as random variables. In addition, it is reasonable to assume that each task execution time is independent because any single HTTP request can be satisfied without any need to process another HTTP request, i.e., each request is self-contained.

Let each task i belong to exactly one class c_i in the set of all task classifications C where membership in a class implies a specific random variable $\underline{T_{c_i j}}$ representing the execution time of that task class on machine j (one of the $\underline{M}$ machines in the HC suite). In this research, it is assumed that the probability distributions describing the random variable $T_{c_i j}$ were created from measurements of the response times of actual requests for data from the site. A typical method for creating such a distribution relies on a histogram estimator [104] that produces a discrete probability distribution known as a probability mass function. Define $\underline{f_{cj}}$ to be a unimodal pmf describing the execution time of tasks in class c on machine j. We consider only unimodal distributions of execution times to simplify the application of the stochastic robustness metric.

Determining the completion time for a machine j, and therefore the completion time of a particular task i, requires a means of combining the execution times for all tasks assigned to that machine. In a deterministic model of task execution times, the estimated execution times for all tasks assigned to machine j would be summed with the machine ready time to produce a completion time. A similar procedure is followed in the stochastic case as well. However, calculating stochastic completion times requires the summation of random variables as opposed to deterministic values. The summation of random variables given their pmfs can be found as the convolution of their corresponding pmfs [63].

Let $\underline{MQ(t)}$ be the set of all tasks that are either pending execution or are currently executing on any of the M machines in the HC suite at time t. To determine the completion time for a task i on machine j at time t, identify the subset of tasks in $MQ(t)$ that were mapped to machine j in advance of task i, denoted $\underline{MQ_{ij}(t)}$. The execution time pmfs for the pending tasks will be convolved [63] with the completion time distribution of the

currently executing task and the execution time distribution for task i on machine j to produce the stochastic completion time pmf for task i on machine j.

The execution time pmf for the currently executing task on machine j requires some additional processing prior to its convolution with the pmfs of the pending tasks to create a completion time pmf. For example, if the currently executing task on machine j began execution at time t_j prior to time t, some of the impulse values of the pmf describing the completion time of the currently executing task may be in the past. Therefore, to accurately describe the completion time of task i at time t requires that these past impulses be removed from the pmf and the remaining distribution renormalized. After renormalization, the resulting distribution describes the completion time of the currently executing task at time t on machine j. To simplify notation, define an operator $\underline{GT(s,d)}$ that accepts a scalar s and a pmf d as input and returns a renormalized probability distribution where all impulse values of the returned distribution are greater than s. The completion time pmf of the currently executing task on machine j is determined by applying the GT operator to its completion time pmf, using the current time t. The resulting distribution is then convolved with the pmfs of the pending tasks on machine j and the execution time distribution of task i to produce the completion time pmf for task i on machine j at the current time t.

8.4 Dynamic Stochastic Robustness Metric (SRM)

8.4.1 Instantaneous SRM

Recall that the individual deadline of each task has been defined in advance; let β_i^{max} denote the deadline for the i^{th} task to arrive to the system. Let $f_{c_1 j}$ be the execution time pmf of the currently executing task on machine j. Order the members of $MQ_{ij}(t)$ according to their scheduled order of execution on machine j and let $f_{c_2 j}$ be the execution time pmf of the first pending task on machine j, with $f_{c_{|MQ_{ij}(t)|j}}$ as the execution time pmf of the last pending task on machine j that is ahead of task i.

Convolution of a scalar with a pmf has the effect of shifting the pmf by the value of the scalar and has no impact on the distribution of probabilities in the pmf. Therefore, if $MQ_{ij}(t) = \emptyset$, i.e., $MQ_{ij}(t)$ is empty, the completion time distribution for task i on machine

j is merely its execution time distribution f_{c_ij} shifted to the current time. The completion time distribution at time t for task i, denoted $\underline{F_i(t)}$ can be found as follows,

$$F_i(t) = \begin{cases} t * f_{c_ij}, & \text{if } MQ_{ij}(t) = \emptyset; \\ GT\{t, t_j * f_{c_1j}\} * f_{c_2j} * \cdots & \\ *f_{c_{|MQ_{ij}(t)|}} * f_{c_ij}, & \text{otherwise.} \end{cases} \tag{54}$$

Following from our prior work on robustness [86] (presented in Chapter 3), the robustness of the finishing time for task i can be found as the probability that task i will finish before its deadline. This probability defines a local robustness characteristic, denoted $\underline{\psi_i(t)}$, and can be expressed as $\mathbb{P}\left[F_i(t) \leq \beta_i^{max}\right]$. The individual local robustness characteristics can then be combined to produce the stochastic robustness metric at time t, denoted $\psi(t)$, as follows,

$$\psi(t) = \prod_{\forall i \in MQ(t)} \left(\mathbb{P}\left[F_i(t) \leq \beta_i^{max}\right] \right). \tag{55}$$

This combination of local robustness characteristics defines an instantaneous measure of robustness (instantaneous SRM) for this resource allocation at a particular time t. Intuitively, the measure defines the probability that all tasks pending or currently executing at time t will meet their deadlines.

8.4.2 Dynamic SRM Value

The instantaneous measure of robustness is used as a basis for defining a single dynamic SRM value. To define that value, recall that a mapping event occurs whenever a task completes execution or a new task arrives at the system. An instantaneous SRM value is generated at each mapping event during the course of a simulation trial. These instantaneous SRM values are then combined to create a sample dynamic SRM value for the resource allocation heuristic. Given the relationship between the dynamic SRM value and the performance metric, the dynamic SRM value can be used to predict the relative performance of two resource allocation heuristics.

If a heuristic consistently maintains a high $\psi(t)$ value over some number of mapping events then there is a consistently low probability that tasks will miss their deadlines over that same period. Therefore, the average $\psi(t)$ value over a large enough number of mapping events should correlate with a consistently low probability that tasks will miss their deadlines. That is, heuristics that maintain a high average $\psi(t)$ value can reasonably be expected to produce a low cost. For this reason, the dynamic SRM value is defined as the average of the instantaneous SRM values found, at each mapping event, during a simulation trial.

8.4.3 Using the Dynamic SRM value

In a dynamic environment, the set of tasks being considered is constantly changing due to task arrivals and completions. Recall that to compute $\psi(t)$ the start time of the currently executing task on each machine is required (i.e., t_j). Determining the start time for a task i requires knowledge of the actual execution time of the previously executed task k on that machine to calculate task k's actual completion time. During simulations used for heuristic evaluation, our methodology utilizes the expectation of the execution time pmf as the actual execution time for each task. Thus, the start time of the subsequent task i is known, enabling the calculation of $\psi(t)$.

Taking the expectation of the class pmfs to produce the most likely actual execution times to use for the evaluation simulations is reasonable in this environment because the task execution time pmfs are assumed to be unimodal. Further study is required to determine the most effective approach when execution time pmfs are not unimodal.

A second factor in evaluating a resource allocation heuristic is the set of tasks and the ordering of task arrivals. Because of variations in the set of tasks to be executed and changes in their arrival ordering, multiple simulation trials should be conducted to adequately predict the typical relative performance among the evaluated resource allocation heuristics. In evaluating our results, we will demonstrate that in a dynamic environment a small number of simulation trials are required to sufficiently indicate the performance of a resource allocation heuristic relative to our given performance objective. Each trial involves

a unique set of tasks with a unique arrival order.

To produce a dynamic SRM value for a resource allocation heuristic a relatively small number of simulations are executed where the mean of each task execution time distribution is used as the actual execution time. This produces a dynamic SRM value for each resource allocation (simulation trial) where the dynamic SRM value is determined as the average of the instantaneous SRM values within that simulation trial. The dynamic SRM values for all of the simulation trials are then combined by taking their average to determine a single dynamic SRM value for the resource allocation heuristic.

The dynamic SRM values for different resource allocation heuristics can then be compared to select the approach that is more robust within the given environment. That is, given the presented formulation of the instantaneous SRM, a simulation trial that has a higher dynamic SRM value should reasonably be expected to produce fewer task deadline misses. In the next section, we present the heuristics that are to be evaluated using the dynamic SRM value in Section 8.7.

8.5 Resource Allocation Heuristics to Evaluate

8.5.1 Introduction

The goal of resource allocation heuristics is to select a mapping of tasks to machines and scheduling of tasks within a machine that minimizes an objective function. The presented heuristics do not directly attempt to maximize the stochastic robustness metric, instead focusing on minimizing the primary objective function. In the results section, the heuristics will be compared using both the stochastic robustness metric and their average performance relative to minimizing Equation 53. All of the heuristics were given a limited amount of time to complete a mapping event.

8.5.2 Two Phase Greedy

The Two Phase Greedy heuristic is based on the principles of the Min-Min algorithm (first presented in [50], and shown to perform well in many environments [26], [66], [87]). The heuristic allocates one task at each iteration, continuing until all task allocations have been resolved. In the first phase of each iteration, the Two Phase Greedy heuristic determines the

best assignment (according to the performance goal) for each of the tasks left unmapped. In the second phase, it selects the task to map based on the best result found in the first phase. The completion time distribution for a given machine j at time t is denoted $\underline{F^j(t)}$. Given the set of tasks assigned to machine j at time t, denoted $\underline{MQ^j(t)}$, $F^j(t)$ can be found as follows:

$$F^j(t) = GT\{t, t_j * f_{c_1 j}\} * f_{c_2 j} * \cdots * f_{c_{|MQ^j(t)|}}. \tag{56}$$

The Two Phase Greedy heuristic is summarized in Figure 31.

while not all tasks are mapped
  for each unmapped task i
    find machine m_j such that
    $m_j \leftarrow \underset{1 \le j \le M}{\text{argmin}} \left[\mathbb{E}[F^j(t) * f_{c_i j}]\right]$;
    resolve ties arbitrarily;
  end for loop
  let A = all (i, m_j) pairs found above
  select pair(s) (x, m_y) such that
  $(x, y) \leftarrow \underset{\forall (i, m_j) \in A}{\text{argmin}} \left[\mathbb{E}[F^{m_j}(t) * f_{c_i m_j}]\right]$;
  resolve ties arbitrarily;
  map x to machine y;
  update $F^y(t)$ based on assignment;
end while loop.

Figure 31: Pseudo-code describing the Two Phase Greedy heuristic.

8.5.3 Segmented Two Phase Greedy (STG)

The segmented two phase greedy heuristic relies on "segmenting" the collection of tasks to be allocated into n groups and then applying a two phase greedy heuristic to each group [107]. A weighting factor is used to determine segments. The new STG heuristic introduced here for this environment calculates a task's weight based on the probability of the task meeting its deadline. That is, the lower the probability that a task will complete within its deadline the higher its queuing priority will be. A weight $\underline{\pi_i}$ that is used to assign the task queueing order is defined for a task i given M machines as follows,

$$\pi_i = \frac{\sum_{j=1}^{M} \mathbb{P}[F_i(t) \le \beta_i^{max}]}{M}. \tag{57}$$

The individual weights are used to define a weighted expected time to compute for each task, denoted $\mathcal{W}_{ij}$ for a given task i on machine j.

$$\mathcal{W}_{ij} = \pi_i \mathbb{E}[T_{c_{ij}}] \tag{58}$$

The weighted expected execution times are combined to produce a weighted expected completion time, denoted $\mathcal{W}_{ij}^{ECT}(t)$ for a given task i on machine j at time t. The weighted expected completion time is calculated using Equation 59.

$$\mathcal{W}_{ij}^{ECT}(t) = \mathcal{W}_{ij} + \mathbb{E}[F^j(t)] \tag{59}$$

Tasks are sorted in ascending order according to the average of their $\mathcal{W}_{ij}$ values across all machines at the time of the mapping event. The sorted task list is then divided into n segments of equal length that are allocated to machines using a two phase greedy heuristic. The two phase greedy heuristic is used to minimize the weighted expected completion time for the last to finish task. Figure 32 presents the details of the STG heuristic discussed here.

```
sort tasks in ascending order by average W_ij value
partition sorted task list into n segments
for each segment S
  while not all tasks are mapped
    for each unmapped task i in S
      find the machine m_j such that
      m_j ← argmin_{1≤j≤M} [W_ij^ECT(t)];
      resolve ties arbitrarily;
    end for loop
    from all (i, m_j) pairs found above
    select pair(s) (x, y) such that
    (x, y) ← argmin_{∀(i,m_j)} [W_{im_j}^ECT(t)];
    resolve ties arbitrarily;
    map task x to machine y;
    update F^y(t) based on assignment;
  end while loop
end for loop.
```

Figure 32: Pseudo-code describing the STG heuristic.

Because a new mapping event occurs each time a task finishes, in general, each machine needs no more than two pending tasks. Thus, once every machine has two tasks pending, the heuristic can terminate the mapping event. This was utilized in the implementation of the STG heuristic to improve the heuristic's execution time.

8.5.4 Negotiation

Iterative approaches have been applied to static mapping problems to search the space of task permutations in a schedule [14,15,102]. Local search also has been applied to *dynamic* scheduling problems [77,82]. The negotiation heuristic introduced here is modeled after such iterative heuristics operating in a dynamic environment. A total ordering of tasks serves as input to a schedule builder that assigns tasks to machines such that the performance objective is maximized. The total ordering is iteratively permuted and the schedule builder re-applied to produce a new resource allocation. Schedules with a higher value are kept, where value is defined by the evaluation of a fitness function. The procedure is analogous to a next-descent search in schedule space.

Each iteration of the negotiation heuristic relies on computing the following two metrics. The local earliness metric quantifies the difference between a task's expected finishing time and its deadline given the current mapping and scheduling. The local earliness metric for a given task i at time t, denoted LEM_{ij}, can be quantified as,

$$LEM_{ij} = \beta_i^{max} - \mathbb{E}[F_i(t)]. \tag{60}$$

Using the local earliness metric the global earliness metric, denoted GEM for a given mapping event, can be found as the sum of the local earliness metrics for all mappable tasks. That is, given a task ordering o and an operator $m(i)$ that returns the machine that task i has been assigned

$$GEM(o) = \sum_{\forall i \in MQ(t)} LEM_{im(i)}. \tag{61}$$

An iteration of the negotiation algorithm is defined as follows. From a current ordering ω of tasks in $MQ(t)$, two tasks are randomly selected for swapping (tasks are initially ordered

by arrival times). The ordering that results from this modification, denoted $\underline{\omega'}$, is used as an activity list for a schedule builder. The schedule builder assigns each task in ω', in order, to the machine that maximizes the local earliness metric as defined by Equation 60. The fitness of the resulting mapping is measured using the global earliness metric defined in Equation 61, where a higher global earliness metric indicates a more fit schedule. If the fitness of this mapping is the best encountered, the ordering ω' is kept for the next iteration. Otherwise, the ordering is discarded and the original ordering ω is maintained for the next iteration. Negotiation terminates after N iterations have been executed. The Negotiation heuristic is summarized in Figure 33.

```
initialize ω to an ordering of all tasks to be mapped
for N iterations
   randomly select two tasks for swapping
   swap ordering of selected tasks
   assign resultant total ordering to ω'
   execute schedule builder using ω'
   if (GEM(ω') < GEM(ω))
      ω ← ω'
end for loop.
```

Figure 33: Pseudo-code describing the Negotiation heuristic.

8.6 Simulation Setup

For this research, it is assumed that 1) the time necessary for a mapping event is negligible compared to the task arrival time, and 2) task execution times are considerably longer than the difference between successive inter-task arrival times.

To evaluate the effectiveness of the dynamic SRM value for predicting heuristic performance we considered two sets of simulations. In the first set of simulations, a small number of simulation trials were conducted to produce a dynamic SRM value for a resource allocation heuristic, as defined in our methodology. For the second set of simulations, a larger number of trials were conducted to produce sample cost values for each heuristic, where the actual execution time for each task is determined by sampling the execution time pmf for that task. For this work 10 simulation trials were used to produce a dynamic SRM value for a resource allocation heuristic as compared to 100 simulation trials to evaluate the

performance metric.

All simulations consisted of 1024 tasks to be processed by eight machines, where task arrival times were not known in advance. Each arriving task belonged to one of five classes whose execution time pmfs for each machine were known in advance. The execution time for a mapping event for all heuristics was limited to 0.1 seconds.

8.7 Simulation Results

The heuristics were compared using the dynamic SRM value and their ability to minimize cost, i.e., the number of tasks that miss their deadlines. Recall that the dynamic SRM value for each heuristic was constructed using a small number of independent simulation trials, where the actual execution times for tasks were set to the expectation of the task execution time pmfs.

Evaluating the success of the dynamic SRM value in comparing heuristics requires the actual performance of each heuristic over a significant number of simulation trials. Given the formulation of the dynamic SRM value, a high dynamic SRM value for a heuristic should indicate a low cost for the heuristic. In other words, heuristics that produce higher dynamic SRM values should have lower cost, i.e., have fewer task deadline misses, than those with low dynamic SRM values.

Figure 34 presents the cost distributions for each of the three heuristics. Each distribution was generated using a kernel density estimator where the kernel function was set to be Gaussian [24]. The cost results from the 100 simulation trials were used as the sample data for the kernel density estimator. Also plotted in the figure are the means of each distribution and the calculated dynamic SRM value for each heuristic. The cost distributions for the three heuristics are only valid in the interval $[0, 1024]$ corresponding to the lowest possible cost for a heuristic and the highest possible cost given the defined simulation setup.

As can be seen in Figure 34, the Negotiation heuristic produced the lowest mean cost of 87.77 and had the highest dynamic SRM value of 0.609. The Two Phase Greedy heuristic had the next lowest mean cost with 279.28 and the next highest dynamic SRM value of

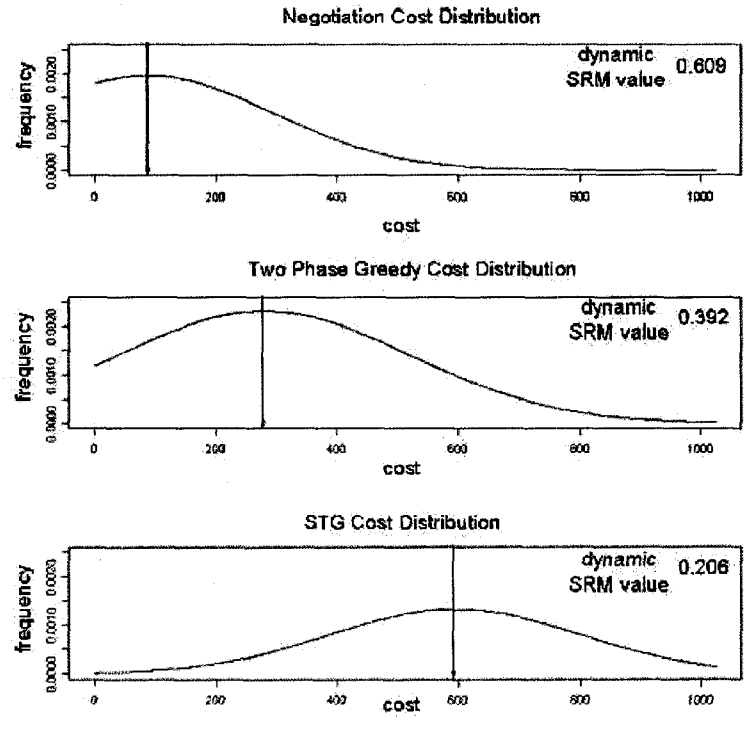


Figure 34: The distributions of cost values for all three heuristics. The cost distributions were generated using a kernel density estimator and the results of the 100 simulation trials.

0.392. Finally, the STG heuristic produced the highest mean cost of 593.6 and the lowest dynamic SRM value of 0.206.

The results of Figure 34 suggest that there is a correlation between the dynamic SRM value and the number of tasks that miss their deadlines. If there is a correlation between the dynamic SRM value and the number of deadline misses, then a plot of all of the simulation sample points taken for the three heuristics should lie on a common curve. To evaluate this conjecture, we combined the sample points for the three heuristics and applied Bayesian regression to produce Figure 35.

The sample costs taken from the three heuristics are plotted as points in the figure where triangles represent the sample points taken from the Negotiation heuristic, diamonds represent the sample points for the STG heuristic, and circles the sample points for the Two Phase Greedy heuristic. Using these data points a Bayesian regression model [24] was generated to fit the points to a common curve. The results of the regression model are plotted in Figure 35 as the the dynamic SRM value versus the logarithm of the cost.

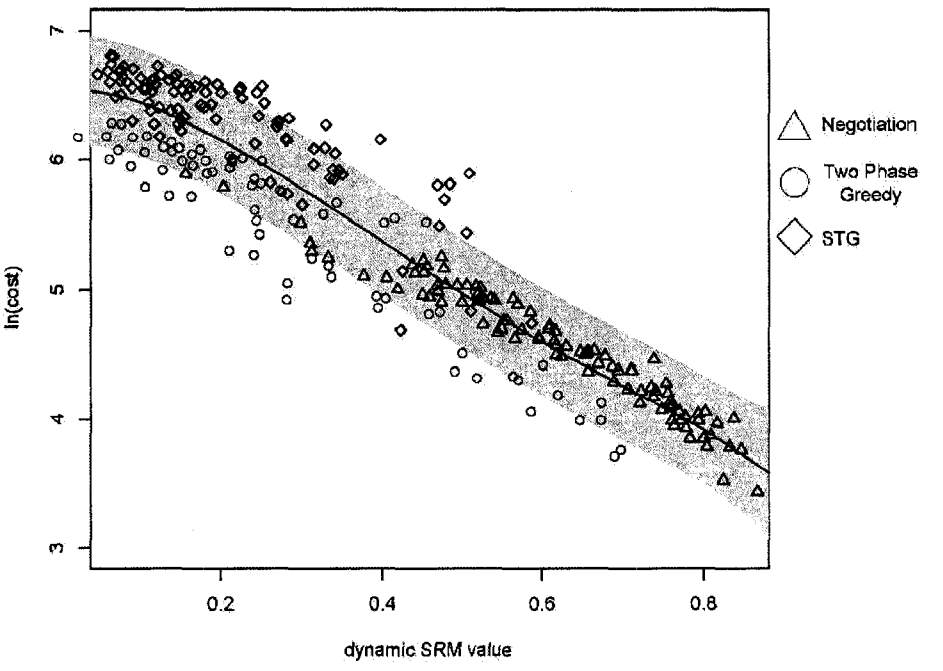


Figure 35: Plot of dynamic SRM value versus the logarithm of the costs for all three heuristics. A Bayesian regression model has been used to fit a curve to the combined set of sample points for all three heuristics. The line in the figure is the mean of the regression model and the shaded region represents one standard deviation around the mean.

The model combines a series of radial basis functions with variance 0.2 that were uniformly distributed on the interval [0,1]. The range of cost results for the simulation trials were re-mapped to the interval [0,1] where 0 corresponds to the smallest possible cost and 1 to the highest possible cost, i.e., 1024. The line plotted in the figure represents the mean of the Bayesian model and the shaded region represents one standard deviation around the mean. The generated model appears to fit the combined sample points taken from the three heuristics to a single simple curve suggesting that there may be a correlation between changes in the dynamic SRM value and changes in cost. Next, we consider the relative performance of the individual heuristics.

From the plot of Figure 34, it appears that the negotiation heuristic generally outperforms the others. To more accurately compare the results of the three heuristics another distribution was generated to assess the frequency with which the Negotiation heuristic outperforms the STG and Two Phase Greedy heuristics. Using the resource allocation cost

data taken from the 100 trials, Figure 36 shows the cost distributions with mean values for both the Negotiation heuristic and the STG heuristic and a cost comparison for the two. To generate the cost comparison distribution, labeled "STG vs. Negotiation Cost Comparison," the cost values generated for the Negotiation heuristic were subtracted from those generated by the STG heuristic for each simulation trial. The samples used to define the actual execution times were drawn in advance of the simulation trials and were the same for each heuristic. A kernel density estimator was then applied to the resultant data points to generate the distributions in the figure. For each simulation trial, the Negotiation heuristic produced a lower cost than that produced by the STG heuristic. However, for a small number of simulation trials the STG heuristic performed comparably to the Negotiation heuristic. These cost comparison values were very close to zero but still slightly positive causing the tail of the cost comparison curve generated by the kernel density estimator to edge into negative values. This suggests that there is a small probability that given the right circumstances STG might perform better than Negotiation.

Figure 37 compares the cost distribution for the Two Phase Greedy heuristic with the cost distribution for the Negotiation heuristic. As can be seen in the comparison plot, labeled "Two Phase Greedy vs. Negotiation Cost Comparison," the density estimate of the comparison cost distribution has a non-zero frequency for a sizable number of negative values. That is, for these trials the Two Phase Greedy Heuristic had fewer task deadline misses than the Negotiation heuristic. However, for the majority of the trials the Negotiation heuristic outperformed the Two Phase Greedy heuristic. From the cost comparison plot this can be seen because the mean of the comparison cost distribution is positive implying that more often than not the Two Phase Greedy heuristic had a higher cost than Negotiation.

8.8 Related Work

In [86], the authors define a stochastic methodology for evaluating the robustness of a resource allocation in a static environment. In that work, uncertainty in system parameters

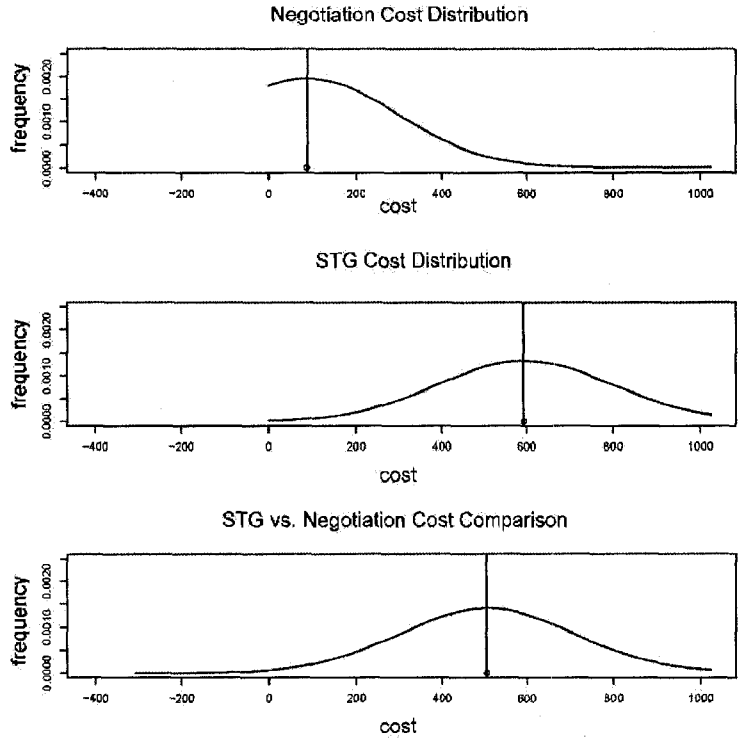


Figure 36: A comparison of the STG heuristic's cost distribution and the Negotiation heuristic's cost distribution. The comparison plot, labeled "STG vs. Negotiation Cost Comparison," shows that the Negotiation heuristic consistently performs better than the STG heuristic for all simulation trials.

and its impact on system performance are modeled stochastically. This stochastic model was then used to derive a quantifiable measure of the robustness of a resource allocation in a static environment. This was done by defining stochastic completion times in a similar manner to our current presentation. A major distinction between the two formulations is that our previous work only considered a static environment where all machines are idle at the beginning of a mapping and the set of all tasks to be mapped is known in advance. In this work, the completion time calculations are very similar but machines may not be idle when a mapping event occurs and new tasks are constantly arriving. The stochastic completion time calculations are an important component of the stochastic robustness metric calculation in [86]. The result of the calculation is a probability distribution for task completion times that is then used in the calculation of the stochastic robustness metric for a resource allocation.

Intuitively, the expression of robustness presented in [86] provides a measure of the

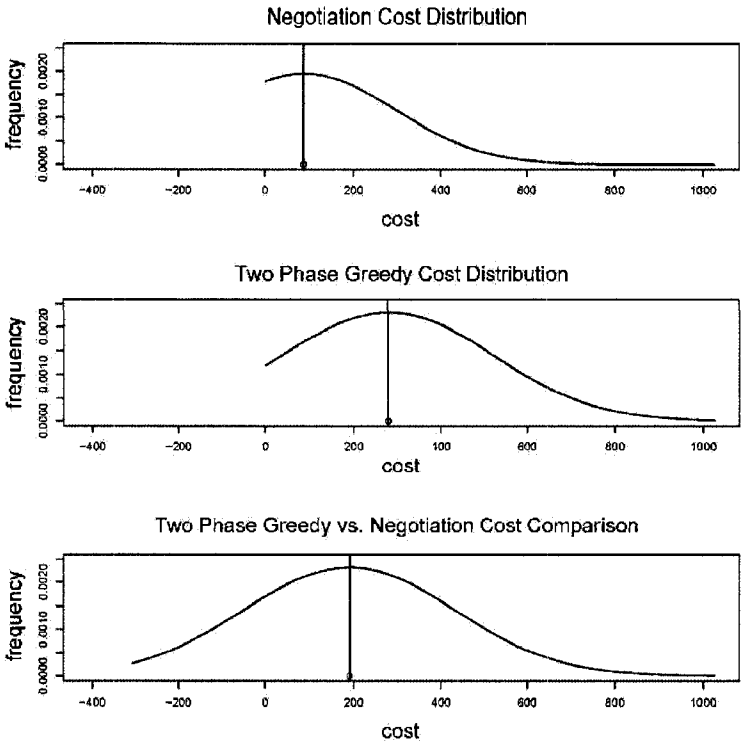


Figure 37: A comparison of the Two Phase Greedy heuristic's cost distribution and the Negotiation heuristic's cost distribution. The comparison plot, labeled "Two Phase Greedy vs. Negotiation Cost Comparison," shows that the Negotiation heuristic was not uniformly better than the Two Phase Greedy heuristic.

likelihood that the makespan of a resource allocation will fall within the provided bounds. This general concept has been used in this work but has been adapted to the details of the present *dynamic* environment. In this work, we were given bounds on the acceptable completion times for each task as opposed to a bound on the acceptable completion time for a collection of tasks. The derived joint probability distribution that defines the stochastic robustness metric corresponds to the probability that all tasks complete by their deadline as opposed to the probability that the collection of tasks will complete by a given deadline as in [86]. A further distinction of this work over [86] is the use of many individual measures of the resource allocation to produce a predictive measure of the robustness.

In [72], the robustness concept is used to develop resource allocations in a dynamic environment. However, that work utilizes a deterministic estimate of task execution times to develop the robustness of the resource allocation's makespan when the task execution time

estimates vary from their predicted values. In this work, the robustness of a given resource allocation heuristic is instead formulated with respect to its ability to meet individual task deadlines. Another distinction between this work and [72] is that task execution times in [72] are simply deterministic execution time estimates. In contrast, this work models task execution times as random variables where we assume the existence of an empirical distribution.

The study in [80] defines a robust schedule in terms of identifying a Partial Order Schedule (POS). A POS is defined as a set of solutions for the scheduling problem that can be compactly represented within a temporal graph. However, the study considers the Resource Constrained Project Scheduling Problem with minimum and maximum time lags, (RCPSP/max), as a reference, which is a different problem domain from the environment considered here.

8.9 Conclusions

Our results suggest that there is an inverse relationship between the dynamic SRM value and the performance objective of the problem studied. In reviewing our results, we explored the general relationship between the dynamic SRM value and the performance objective by analyzing the fit of a Bayesian regression model to our results. The relatively good fit of the regression model to the combined data of the three heuristics is strong evidence of the relationship between the dynamic SRM value and the stated performance objective. Our results appear to demonstrate that the dynamic SRM value can simplify the evaluation of resource allocation heuristics in a dynamic environment. That is, the methodology for determining the dynamic SRM value for a heuristic reduces the number of simulations required to demonstrate the superiority of one heuristic over another in a dynamic resource allocation environment.

Using these results we compared the performance of three different heuristics taken from the literature and applied to a stochastic dynamic environment. From this comparison, the Negotiation heuristic showed promising results in this environment. Given the apparent

relationship between the dynamic SRM value and the stated performance objective, a valuable extension of this work would be the development of resource allocation heuristics that incorporate the dynamic stochastic robustness metric during a resource allocation.

As an extension of this initial work, we will consider the application of the stochastic robustness concept within a resource allocation heuristic. This extension will explore how best to use the presented metric within a resource allocation. An important element that must be investigated within this context is how to differentiate between two resource allocation events that both present 0 probability for *all* tasks to meet their completion time goals. One proposal for accommodating this lack of fidelity in the robustness metric is to identify all of the tasks that have a 0 probability to meet their completion time goals and consider the robustness metric for only the remaining tasks. By adding this dimension to the robustness metric we can increase the sensitivity of the metric in over-subscribed situations that are of principal interest. Our proposal is to evaluate the use of the dynamic robustness metric during resource allocation and possibly extending the definition of the metric to improve its capabilities within a resource allocation.

CHAPTER 9

BAYESIAN INFERENCE BASED RESOURCE ALLOCATION IN A HETEROGENEOUS COMPUTING SYSTEM

9.1 Introduction

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

Often, heterogeneous, distributed computing systems must operate in environments where uncertainty in system parameters is common. Robustness in this context can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed [3]. We define a stochastic robustness metric [89, 93] for quantifying the robustness of a resource allocation in a stochastic *dynamic* environment. In this environment, the exact execution time of any given application is dependent on the details of the image that is to be processed. Thus, the execution times of these applications may be highly variable and are treated as random variables. Because the list of algorithms that may be requested is limited, the execution time random variable for each algorithm is assumed to be well characterized. That is, we assume that a probability mass function (pmf) is available for each application execution time random variable [104] (determined by either experimental or analytical techniques [64]).

The exact sequence of user requests for processing are unknown prior to their arrival,

i.e., job arrival times are not known in advance. Each arriving request is assigned a deadline relative to its arrival time, the size of the image that is to be processed, and the expected execution time of the algorithm requested (averaged across all machines). The system provider requires that each request complete by its assigned deadline. However, if the system were to fail to complete a request by its deadline, then the request must be completed on a best effort basis. From this set of requirements, we formulate a robustness metric for resource allocations in terms of the probability that the allocation will complete all assigned requests by their deadlines. We use this formulation of the stochastic robustness metric to design resource allocation heuristics capable of allocating a dynamically arriving set of requests to a heterogeneous computing system. The problem of resource allocation in the field of heterogeneous parallel and distributed computing is NP-complete (e.g., [32, 50]), therefore, the development of heuristic techniques to find near-optimal solutions represents a large body of research (e.g., [1, 36, 39, 42, 50, 61, 66, 103]).

The major contributions of this work include: (1) a mathematical model for quantifying the robustness of resource allocations in a stochastic dynamic environment that can be used *during* resource allocation to inform decision making and (2) the design of two novel resource allocation techniques based on this formulation of robustness. Specifically, we propose a one-step lookahead procedure for resource allocation that compares the value of available allocation decisions for their impact on the defined robustness metric. Our results clearly suggest the viability of this approach in this environment.

In the next section, we present an overview of the system model used to evaluate the chosen approach. Section 9.3 presents our mathematical model of robustness in a dynamic environment. Two new heuristics based on this model of robustness are presented in Section 9.4. The details of the simulation setup used to evaluate our heuristics are presented in Section 9.5. Section 9.6 provides the results of our simulation study. A sampling of the related work is presented in Section 9.7 and Section 9.8 concludes the paper.

9.2 System

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

The set of applications to be executed is assumed to be composed of a collection of frequently run algorithms. The actual execution time of each application is dependent on the data that are to be processed, where the exact details of this dependence are not known in advance. However, we assume that an accurate pmf describing the possible execution times for each application exists and is available to the resource manager to aid in decision making. There are several existing techniques for generating such a probability distribution [51].

The system is required to complete each request for processing by its assigned deadline. If the system fails to complete a request by its assigned deadline, then it is penalized a fixed amount for each failed request. For any requests that miss their assigned deadline, the system is required to complete them on a "best effort" basis, i.e., requests must be completed as soon as possible when they miss their deadline. The goal of resource allocation heuristics in this environment is to minimize the number of requests that miss their deadline.

9.3 Mathematical Model

9.3.1 Stochastic Application Completion Time

Although some applications may belong to the same class, application execution times are inherently data dependent and may vary in an unpredictable manner. Application execution

times are modeled as random variables.

We assume that all applications that are to be executed by the system can be individually classified into one of $\underline{C}$ classes according to the details of the application that is to be executed. That is, each application i is assigned to exactly one class, denoted $\underline{c_i}$, ($c_i \in C$). The execution time of each application i ($i \in c_i$), executed on machine j (one of the $\underline{M}$ machines in the HC suite), is modeled as a random variable, denoted $a_{c_i j}$. In addition, each application execution time is assumed independent, i.e., there are no inter-application data dependencies in this environment. This assumption of independence is valid for non-multitasking execution mode which is commonly considered in the literature [36,59,66,103]), and applied in practice in a variety of systems, e.g., an iterative universal datagram protocol server model [43].

We assume that the probability distribution describing the random variable $a_{c_i j}$ has been created from measurements of the response times of actual application executions. A typical method for creating such distributions relies on a histogram estimator [104] that produces a discrete probability distribution known as a probability mass function (pmf). Each application class defines a set of pmfs that each describe the probability of all possible execution times for that class of application on each machine within the HC suite. The pmf describing the execution time of application class c executed on machine j is denoted $\underline{\tau_{ij}}$. We assume that the collection of application execution time pmfs have been provided in advance and that all of the application classes that the system may be asked to execute are known prior to system execution. Finally, each arriving application i is assigned a deadline for completion, denoted $\underline{\delta_i}$, based on its arrival time, denoted $\underline{\alpha_i}$.

Determining the completion time for a machine j at time-step $t^{(k)}$, and therefore the completion time of a particular application i, requires a means of combining the execution times for all applications assigned to that machine. Using a deterministic model of task execution times, the estimated execution times for all applications assigned to machine j would be summed with the machine ready time to produce a completion time. A similar procedure is followed using a stochastic model as well. However, calculating stochastic

completion times requires the summation of random variables as opposed to deterministic values. A summation of random variables can be found as the convolution of their corresponding pmfs [63, 78].

Let $\underline{MQ(t^{(k)})}$ be the set of all applications that are either pending execution or are currently executing on any of the M machines in the HC suite at time-step $t^{(k)}$. To determine the completion time for an application i on machine j at time-step $t^{(k)}$, identify the subset of applications in $MQ(t^{(k)})$ that were assigned to machine j in advance of application i that have not yet completed execution, denoted $\underline{MQ_{ij}(t^{(k)})}$. The execution time pmfs for the pending applications will be convolved [63, 78] with the completion time distribution of the currently executing application and the execution time distribution for application i on machine j to produce the stochastic completion time pmf for application i on machine j.

The execution time pmf for the currently executing application on machine j requires some additional processing prior to its convolution with the pmfs of the pending applications to create a completion time pmf. For example, if the currently executing application on machine j began execution at time-step $t^{(j)}$ $(j < k)$, some of the impulse values of the pmf describing the completion time of the currently executing application may be in the past. Therefore, to accurately describe the completion time of application i at time $t^{(k)}$ requires that these past impulses be removed from the pmf and the remaining distribution renormalized. After renormalization, the resulting distribution describes the completion time of the currently executing application at time-step $t^{(k)}$ on machine j. To simplify notation, define an operator $\underline{GT(s,d)}$ that accepts a scalar s and a pmf d as input and returns a renormalized probability distribution where all impulse values of the returned distribution are greater than s. The completion time pmf of the currently executing application on machine j is determined by applying the GT operator to its completion time pmf, using the current time-step $t^{(k)}$. The resulting distribution is then convolved with the pmfs of the pending applications assigned to machine j and the execution time distribution of application i to produce the completion time pmf for application i on machine j at the current time-step $t^{(k)}$.

9.3.2 Calculating Robustness

The robustness of a resource allocation in this environment is defined by the joint probability that all applications will complete by their assigned deadline at a given time-step $t^{(k)}$. To calculate this value, for each machine j, we calculate the joint probability of completing all applications assigned to this machine by iteratively applying Bayes theorem [24] to convert the joint probability of completing all applications by their deadline into a combination of simpler known probabilities. The basis for this calculation is the known probability of completing the currently executing application a_{1j} by its deadline, denoted $\underline{p(a_{1j})}$. Because the start time of the currently executing application is known and its completion time distribution is not dependent on any of the remaining applications assigned to this machine, we can find this probability directly. For machine j, we calculate the completion time distribution of a_{1j} by convolving its execution time distribution with its start time comparing the completion time distribution for a_{1j} with its deadline to find $p(a_{1j})$. Given n_j applications assigned to machine j, we iteratively apply Bayes theorem as follows:

$$\begin{aligned} p(a_{2j}) &= p(a_{1j})p(a_{2j}|a_{1j}) \\ p(a_{3j}) &= p(a_{2j})p(a_{3j}|a_{2j}) \\ &\cdots \\ p(a_{n_jj}) &= p(a_{n_j-1j})p(a_{n_jj}|a_{n_j-1j}). \end{aligned}$$

Convolving the distribution of $p(a_{1j})$ with the *execution* time distribution of a_{2j} gives the *completion* time distribution for a_{2j}, given a_{1j} completed by its deadline. The final step in determining $p(a_{2j}|a_{1j})$ is to compare the completion time distribution for a_{2j} with its deadline. That is, $p(a_{2j}|a_{1j})$ is found as the sum over the completion time distribution for a_{2j} that corresponds to a_{2j} completing by its deadline. More generally, to calculate each $p(a_{ij}|a_{i-1j})$, we extract the portion of the completion time distribution for a_{i-1j} whose completion times are less than or equal to the deadline for a_{i-1j}. This portion of the

probability distribution is then renormalized to form the pmf corresponding to $p(a_{(i-1)j})$.

At any time-step $t^{(k)}$, each application in the resource allocation has been previously assigned to the input queue of some machine j ($1 \leq j \leq M$). Recall that there are no explicit inter-application dependencies—the only dependence that may exist between applications is their reliance on the same machine for execution. Thus, we can formally define the stochastic robustness of a resource allocation at a given time-step $t^{(k)}$, denoted $\theta^{(k)}$, as the product of each joint probability associated with a given machine. That is,

$$\theta^{(k)} = \prod_{\forall j} p(a_{1j}, a_{2j}, \cdots, a_{n_j j}). \tag{62}$$

9.4 Heuristics

9.4.1 Immediate Bayes Allocator

Finding an optimal allocation in this environment requires the evaluation of each possible application allocation at every time-step throughout the entire simulation [20]. Unfortunately, for any problem of reasonable size, this calculation is infeasible to complete within the context of a dynamic system such as ours. We explore an approximation of an optimal allocation policy that constrains the set of controls considered at each time-step to a small subset of those available.

We begin our approximation by sorting the available applications based on the urgency of each request.Urgency is determined based on a combination of the deadline for completing the request, its arrival time, and average expected execution time of the application associated with the request. That is, the urgency of the k^{th} application request, denoted u_k, can be calculated as,

$$u_k = \delta_k - \alpha_k - 1/M \sum_j \left(\mathbb{E}[\tau_{c_k} j]\right). \tag{63}$$

Each allocation begins by ranking all pending requests based on urgency and sorts them according to decreasing urgency. For the most urgent task, we identify the machine that provides the highest probability to complete the application before its deadline. Because

the robustness values are limited to the range [0,1], multiple machines may equivalently maximize robustness. In this case, the heuristic identifies all of the machines that would achieve this robustness value for the task allocation. In addition to finding the best allocations at this time-step, the heuristic also identifies the best allocations one time-step in the future. The probabilities associated with these two choices are then compared to determine if there is any penalty for waiting to assign the request now. That is, as long as there is no reduction in robustness in the next time-step relative to the robustness value achievable in this time-step, the heuristic will wait on assigning the request. By waiting, the heuristic may benefit from the additional information that is revealed in the next time-step, e.g., an application may complete earlier than expected allowing for a more robust assignment on another machine.

If the heuristic identifies that the current robustness value is higher than the robustness value in the next time step, then the heuristic is at a critical point in the allocation of the request under consideration. That is, the probability of completing the selected request by its deadline is declining from this time-step to the next. Once a critical point has been reached, the heuristic assigns the request to its chosen machine in this time-step to avoid the obvious decline in the robustness of the resource allocation.

Alternatively, if the most urgent request can wait an additional time-step with no degradation in robustness, then the heuristic checks the remaining unmapped requests. Each unmapped request is considered for allocation in order of urgency, identifying the machine assignments for the request that maximize the probability of completing the request by its deadline, in both this time-step and in the next. If the robustness of the completion time for the chosen assignment is lower in the next time-step than in this time-step and the selected machine is different from the chosen machine for the most urgent request, then we assign the request. However, if there is a change in robustness but the selected machine is the same as the selected machine for the most urgent request, then we apply the sufferage concept of [66] comparing the difference in robustness values for the best and second-best machines for each of these requests. If the most urgent request would suffer the most by not getting its best machine, then the heuristic waits on the application assignment. Otherwise, the

alternate application is assigned to its first choice machine over the most urgent request. That is, if the most urgent request would suffer the most, then we know that we can wait another time-step and achieve the same robustness value for this request. Thus, we can avoid committing to any assignments at this time-step. If there is no penalty for waiting on the second most urgent request, then we proceed to examine all other remaining requests in order of urgency using this same procedure.

Allocations based on deadlines ignore the arrival process for new requests. For example, assume that a number of requests have recently arrived that all have relatively loose deadlines, i.e., none of the requests will be allocated at this time-step or in the near future. Next, in a future time-step a number of requests arrive that have tight deadlines. By this time, the robustness for the earlier requests may begin to decline. That is, in this time-step all of the pending requests may need to be assigned, but the system does not have sufficient capacity to complete all requests in such a short period of time. However, by allocating the original requests when they arrived, the heuristic may have been able to avoid this artificial congestion. Our simple approach accounts for this situation by limiting the number of pending requests at each time-step. In the event that the number of pending requests exceeds this threshold, then the heuristic assigns the most urgent request to the machine that maximizes the probability of completing the request by its deadline, continuing in this manner until the number of applications drops below the provided threshold. For our environment, the best results were achieved using a threshold of 5 pending applications.

9.4.2 MaxRobust

The MaxRobust heuristic applies a simple greedy approach to maximizing the robustness of the resource allocation. Upon the arrival of each new application execution request, MaxRobust assigns the application to the machine that maximizes $\theta^{(k)}$ at each time-step $t^{(k)}$. That is, MaxRobust calculates the $\theta^{(k)}$ value for each possible machine assignment selecting the assignment that maximizes $\theta^{(k)}$. In this way, MaxRobust greedily assigns applications so as to attempt to maximize the joint probability that all applications complete by their deadline.

9.4.3 Minimum Completion Time (MCT)

For comparison, we implemented a Minimum Completion Time (MCT) heuristic [66, 110] that ignores the robustness of each allocation, instead allocating applications such that their expected completion time is minimized. That is, upon arrival each application is assigned to the machine that provides the earliest expected completion time.

9.5 Simulation Setup

Our simulation environment consisted of eight machines that exhibited inconsistent heterogeneous performance. To model the sample mean application execution times for each class of application on each machine, we used the base execution results for the twelve SPECint 2006 benchmark applications [98]. The execution times of the selected benchmarks were used to define the mean of a gamma distribution used to generate 500 random sample execution times for each application class on each machine. After generating the sample execution times, we applied a histogram to the result to produce a noisy approximation of the original distributions—one for each application class assigned to each machine. Each benchmark served as a model for an application class to be executed by the system creating an eight by twelve matrix of execution time pmfs.

To evaluate the effectiveness of our heuristics we conducted 50 different simulation trials. Each simulation modeled a bursty arrival period of application requests. That is, the frequency of application arrivals during each simulation is such that none of the heuristics tried were able to complete all applications by their assigned deadline for each trial. Each simulation trial included 150 applications that arrived over a period of 1,900 time-steps. Application arrivals were assumed to follow a Poisson process, where the class of the arriving application was randomly selected with a uniform probability.

9.6 Results

The heuristics were evaluated based on the percentage of the arriving applications that completed by their assigned deadlines. Figure 38 shows a plot of the results for our heuristics along with their 95% confidence intervals. From the results of our simulation trials, both

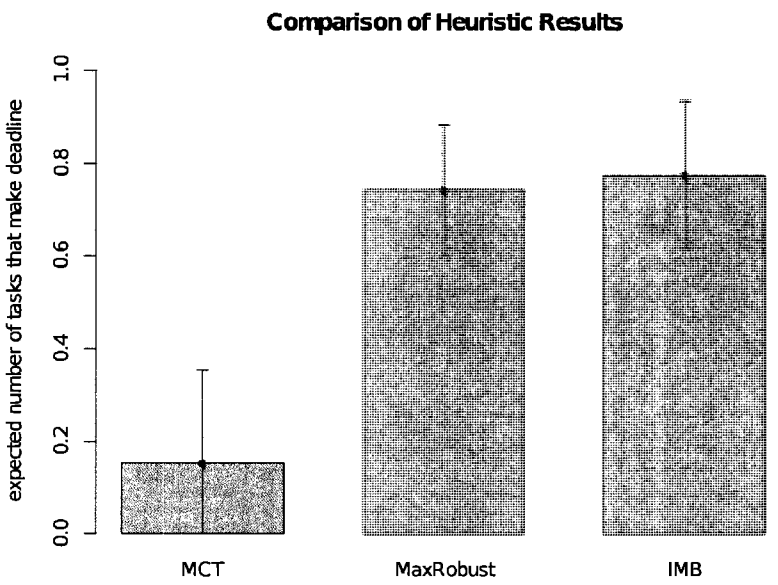


Figure 38: A comparison of our heuristic results over 50 simulation trials. The results achieved for each heuristic are plotted along with their 95% confidence intervals.

MaxRobust and the Immediate Bayes Allocator are capable of outperforming the naive MCT approach. In our simulations, the MaxRobust and MCT heuristics both operate in an immediate mode, i.e., applications are assigned to machines as soon as they arrive. However, the MaxRobust approach makes allocation decisions that only maximize robustness while MCT allocation decisions attempt to only minimize application completion times. The performance difference between these two approaches is significant and suggests the value of the stochastic robustness model for making immediate allocation decisions.

The IMB heuristic on average slightly outperforms the MaxRobust heuristic both in the number of applications that make their deadline and the average delay incurred by each application that misses its assigned deadline. Both heuristics make allocation decisions based on robustness, but the IMB heuristic also incorporates the value of information into resource management. That is, the IMB heuristic generally waits to allocate applications until any further waiting would decrease the probability of an application making its assigned deadline.

Figure 39 compares the three heuristics on the basis of the number of simulation trials

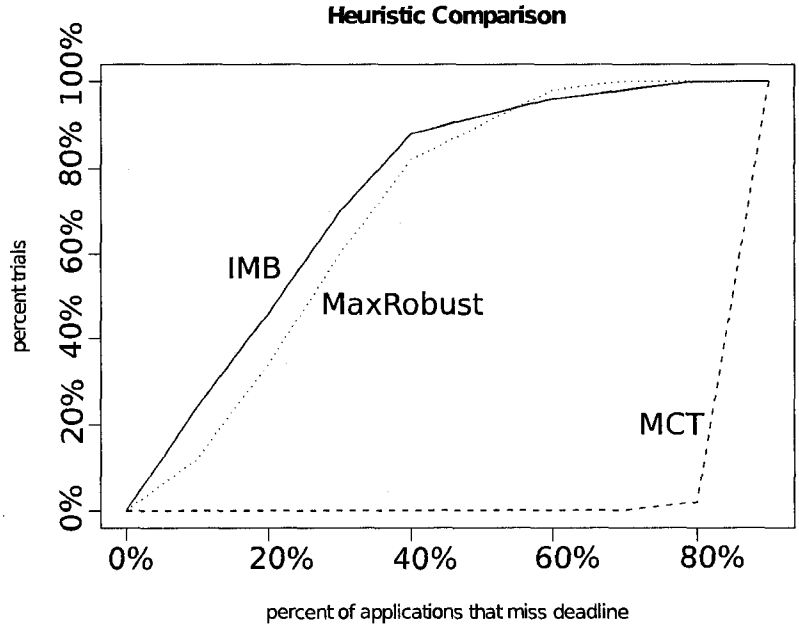


Figure 39: A comparison of the three heuristic results, where each point on the curve corresponds to the percentage of applications that miss their deadlines (plotted on the x-axis) relative to the percentage of trials where this occurs (plotted on the y-axis). The solid line corresponds to the results for the IMB heuristic, the dotted line corresponds to the results for the MaxRobust heuristic, and the dashed line corresponds to the results for the MCT heuristic.

where a given percentage of the applications miss their deadlines. The x-axis of the figure defines the percentage of applications that miss their deadlines and the y-axis of the figure expresses the corresponding percentage of the simulation trials. The best result achievable in the figure corresponds to the line $y = 1$ (i.e., no applications ever miss their deadline) and the worst result is defined by $y = 0$ (i.e., all applications miss their deadlines). Comparing our heuristic results to these two known limits on performance, we see that the MCT heuristic (plotted as a dashed line in the figure) appears generally incapable of completing applications by their assigned deadlines. We also see that the IMB heuristic (solid line) generally outperforms the MaxRobust heuristic (dotted line).

Finally, we compare the IMB and MaxRobust heuristics in terms of average completion time delay. We define average completion time delay as the average additional time that needs to be added to the deadline of each request to ensure it would complete on-time.

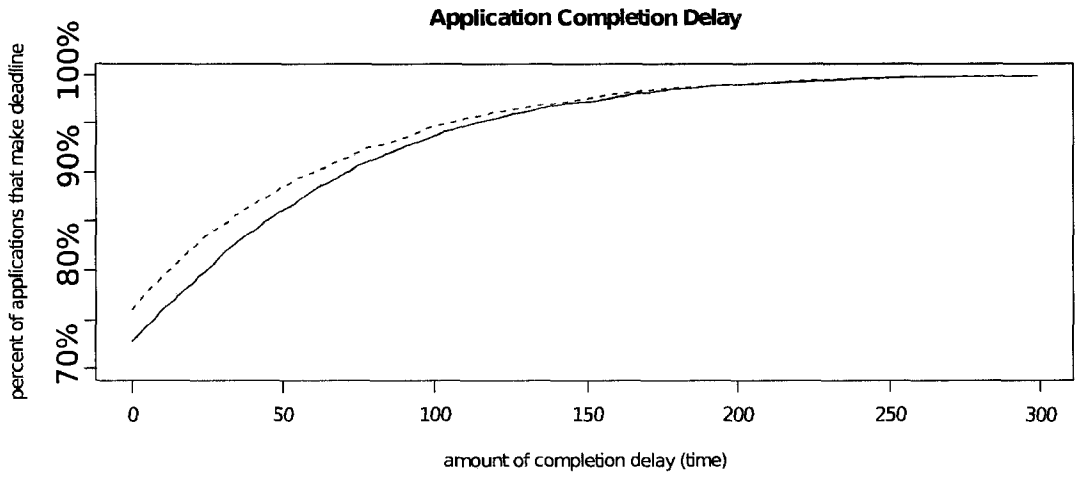


Figure 40: Average completion time delay versus the percentage of applications that complete by their adjusted deadline for the MaxRobust heuristic (plotted as a solid line) and the IMB heuristic (plotted as a dashed line).

Using this adjusted deadline, we can compare the simulation results of the heuristics based on the percentage of requests that complete by their adjusted deadline. We range the adjusted deadline from 0 to 300. Increasing, application deadlines by 300 enabled both heuristics to complete all applications on-time. Figure 40 provides a comparison of average completion time delay for the MaxRobust (plotted as a solid line in the figure) and IMB (plotted as a dashed line) heuristics. In the figure, the x-axis defines the amount of time that each deadline would have to be increased to ensure that the corresponding percentage of requests identified on the y-axis would complete on-time. In this figure, the best values correspond to the smallest increase in deadline (small x-axis values) that result in the largest percentage of applications that complete on-time (large y-axis values). Recall that for this environment, the heuristics are required to complete all applications on a best effort basis, if they miss their assigned deadlines. We define "best effort" to mean as close to the assigned deadline as possible. From the results of Figure 40, we can see that for the majority of applications that miss their deadline the IMB heuristic finishes more of these applications closer to their assigned deadline than the MaxRobust heuristic. The difference in average completion delay between the two heuristics may account for the difference in overall performance seen in the results of Figure 39.

9.7 Related Work

According to the literature, the problem of workload distribution considered in our research falls into the category of dynamic resource allocation, assuming that multiple invocations of a resource allocation heuristic are overlapped in time with task arrivals. The general problem of dynamically allocating a class of independent tasks to heterogeneous computing systems was studied in [66]. The primary objective in [66] was to minimize system makespan, i.e., the total time required to complete all tasks sent for mapping. This objective is very different from the primary objective in our work: complete each application execution before its assigned deadline. Both our MaxRobust and IMB heuristics employ our robustness model to effectively achieve this goal. For comparison, we employed the MCT heuristic of [66] in this environment and demonstrated the superior performance of the MaxRobust heuristic at achieving our resource allocation objective. The research in [66] assumes no deviation of the actual time to compute a task from its estimated time to compute (ETC) value, i.e., the performance predicted by a resource allocation heuristic is assumed to match the actual performance. In our environment, the uncertainty in application execution times is significant and must be accounted for in the mathematical model.

The robustness requirement in this work differs substantially from our earlier work on robustness in a dynamic environment [72]. In [72], the robustness requirement was expressed in terms of the overall resource allocation, i.e., expressed in terms of the entire allocation. In this work, each application has an individual deadline, thus, the robustness metric must be expressed in terms of individual applications. In reviewing our results, we compare our heuristics using a robustness metric that combines the pending application completion time distributions into an evaluation of heuristic performance at each time-step.

In [93], similar to this environment, each dynamically arriving task is assigned its own deadline relative to its arrival time and the execution time of each task is modeled as a random variable. However, our previous work focused on predicting the performance of heuristics in a stochastic dynamic environment. In this work, we model robustness in a manner that enables the use of the resulting robustness metric during resource allocation. In addition, this work presents two novel resource allocation techniques based on our robustness

metric that appear to perform well in this environment.

The study in [80] defines a robust schedule in terms of identifying a Partial Order Schedule (POS). A POS is defined as a set of solutions for the scheduling problem that can be compactly represented within a temporal graph. However, the study considers the Resource Constrained Project Scheduling Problem with minimum and maximum time lags, (RCPSP/max), as a reference, which is a different problem domain from the environment considered here.

9.8 Conclusions

In this work, we defined a model of stochastic robustness that facilitates its use during resource allocation. We applied this model of robustness to the design of two novel resource allocation heuristics capable of assigning applications to machines in a manner that minimizes the number of applications that miss their deadline. Both heuristics showed significant promise for achieving the desired result in a dynamic environment. In this research, it appeared that the MaxRobust heuristic was comparable to the IMB heurisitic and was significantly less complex. A detailed comparison of our simulation results for the MaxRobust and IMB heuristics suggests that in this environment there may be limited value to new information acquired during resource allocation. The results clearly show the value of our robustness model in aiding resource allocation decision making. Future work in this area should investigate the application of the IMB heuristic to environments where application execution times are less predictable. In such an environment, actual application completion time information should be more valuable

CHAPTER 10

OVERLAY NETWORK RESOURCE ALLOCATION USING A DECENTRALIZED MARKET-BASED APPROACH

10.1 Introduction

Recently, information technology systems have begun to rely heavily on the concept of Services Oriented Architecture (SOA). SOA is a means of leveraging existing applications as services within a distributed computing environment to develop new applications. One mechanism commonly used to integrate existing applications is known as the enterprise services bus (ESB) [83]. According to the Business Integration Journal [38], "The [ESB] supports the unifying integration infrastructure required for SOA and heterogeneous environments." By relying on an ESB to implement a distributed application, service requesters—using the ESB to communicate with service providers—need not depend on the details of service provider implementations, e.g., the physical location of the service provider. Instead, service requesters depend on the abstract definition of the service that they are using and trust the ESB to forward their requests to an appropriate service provider. Because the service requester is not dependent on an explicit instance of a service provider, multiple service providers could be deployed within the ESB to provide additional capacity for a service that is in high demand.

An important aspect of an ESB implementation is that it can be decentralized both to increase its reliability and to ensure its scalability [23,38]. A common approach to increasing the reliability of a system is to duplicate that system many times across many hardware deployments, a technique often referred to as replication [23,33].

Successfully replicating ESB components requires maintaining network transparency

---

A preliminary version of this research appeared in [94].

[33]; i.e., the user of the ESB infrastructure should be shielded from the existence of any redundant components used to provide the replicated ESB or its attached services. To the user, the ESB should appear as a single highly available system that always has sufficient capacity to route service requests. Achieving this kind of transparency in service delivery requires a mechanism for allowing the ESB infrastructure to adapt dynamically to changes in system load. That is, transparently utilizing replicas within an ESB infrastructure requires that the replication infrastructure provide a mechanism for routing requests to their physical destination. In this work, we focus on the allocation of ESB resources to the shipment of service requests to service providers.

We investigate the application of a decentralized market-based approach to resource allocation within a heterogeneous deployment of a replicated ESB environment. Service requesters send tasks to service providers using an overlay network provided by the ESB infrastructure. The decision of how to allocate ESB capacity to service requests is made in a decentralized manner based on a quantification of current resource demand relative to current system capacity. Individual service requesters select transmission rates through ESB resources that maximize their individual benefit, while the ESB adjusts "prices" for network links and ESB computing capacity to reflect current demand. Thus, price setting enables service requester decision making by enabling shared resources to communicate a simple quantification of current system congestion to requesters.

Figure 41 presents a graphical model of a simple overlay network provided by a replicated ESB. Service requesters are depicted in the figure as triangles, service providers as squares, and ESB components as circles. Each service requester is connected to a collection of ESB components that "service" requests by dispatching them to a service provider capable of completing the request. The number of requests produced by each service requester may vary with time according to some unknown process. The capacity of the ESB components to service incoming requests may differ from one component to another, i.e., the collection of ESB components are assumed heterogeneous in their performance [39, 45, 56]. Finally, each ESB component is connected to a collection of service providers by a finite capacity link.

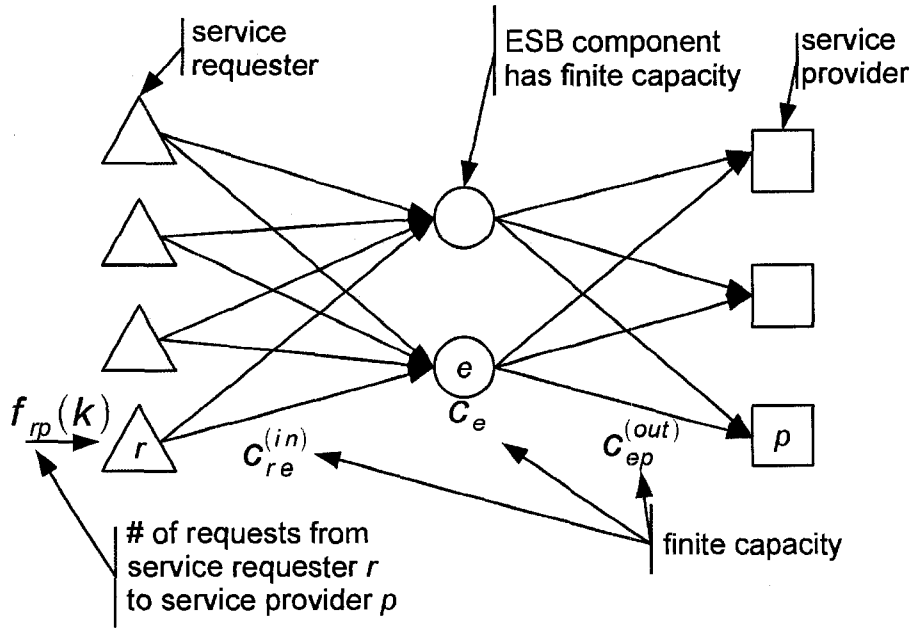


Figure 41: An example system where four service requesters are utilizing two ESB components to communicate with three service providers. Service requesters are shown as triangles, ESB components as circles, and service providers as squares. Each input link to an ESB component from a service requester has a finite capacity, denoted $c_{re}^{(\text{in})}$. Likewise, each output link from an ESB component to a service provider has a finite capacity denoted $c_{ep}^{(\text{out})}$. Finally, each ESB component has a finite capacity for servicing requests, denoted c_e.

Our mechanism for resource allocation can be thought of as a market-based approach where market demand for shared resources helps the system to set prices for those resources. Some market-based approaches rely on an auction to create a market where prices are set by the highest bidder [11, 27, 41, 60]. In contrast, our approach utilizes price setting based on duality theory [30]. In duality theory, selected constraints are directly accounted for in the optimization criterion as a penalty. This approach is analogous to that used in congestion control on the Internet [99] and in ad-hoc sensor networks [29, 31, 68]. In our system model, price variables are introduced to model market demand for shared resources, where prices provide a simple quantification of demand relative to supply. For example, as the number of requests through an ESB component increases, the ESB component raises its "price." Conversely, if the number of requests through an ESB component decreases, its quoted price also decreases. In addition to measuring the demand for the component

itself, the ESB component also is responsible for stating the demand for the links from that ESB component to all of the service providers with which the component can communicate. The procedure used to calculate optimal prices for resources is presented in Section 10.3. Individual service requesters directly utilize the current pricing information provided by the ESB components to make local resource allocation decisions about how best to assign their volume of requests within the overlay network given current network utilization. In concert, we will show that this mechanism results in provably optimal resource allocations.

In the replicated ESB environment, we assume that each service provider offers a unique service to the overall system and that each service provider always has sufficient capacity to service all incoming requests. In a real system, a service provider may be implemented using a collection of finite-capacity replica providers, where the replicas combine to provide a more reliable scalable service implementation. Within each of these collections, we treat the allocation of finite-capacity service provider resources as a separate resource allocation problem. In Figure 42, we have re-drawn the distributed ESB environment to show how the collection can be used to provide such a service to the ESB system. When treated in isolation, the allocation of service provider capacity within each collection is simpler than our original distributed ESB problem because there is only one class of shared resource to manage.

To demonstrate resource allocation in this simpler environment, we apply it to a related distributed web hosting environment [8]. In a distributed web hosting environment, the hosted web site is replicated to multiple web servers to increase the apparent reliability and performance of the web site. Incoming user-driven HTTP requests for data are routed to the web servers for processing by a collection of independent service requesters. By indirecting web server access through service requesters, we can allocate incoming user requests using a decentralized approach that is similar to that of the replicated ESB environment. The overlay network topology of this environment includes only service requesters and service providers, where a service provider is defined to be a web server. This two-layer overlay network can also be used to model the allocation of requests by ESB components to replicated service providers in our previous example. That is, in this simpler model, the ESB

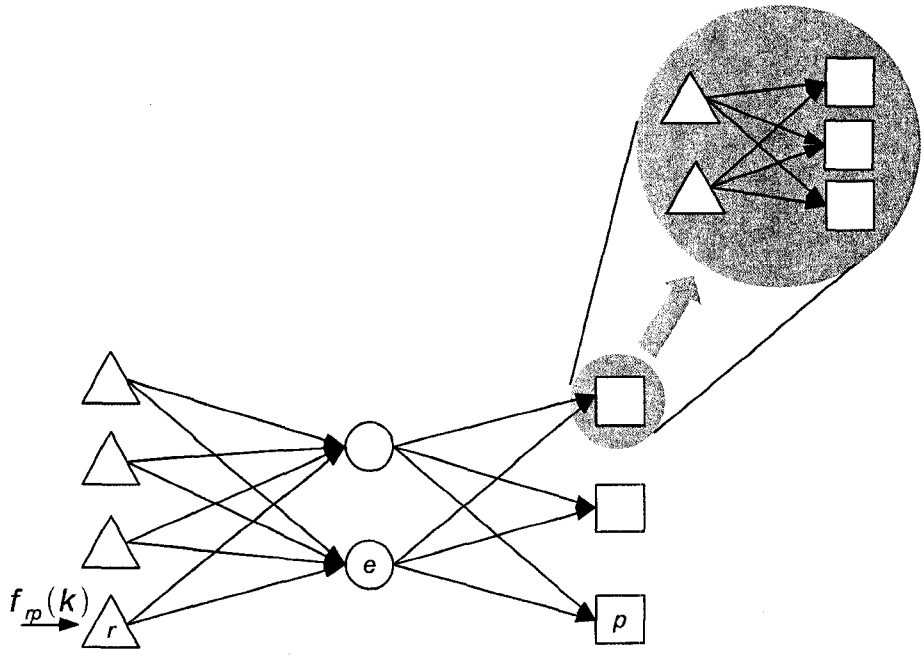


Figure 42: An example use of finite-capacity service provider resources to implement a scalable reliable service within the context of the distributed ESB system. We have expanded the depiction of a service provider to reflect the added complexity of providing a scalable, highly available implementation through replication. In this example, the ESB environment operates as before and the added complexity within a service provider can be treated as a separate allocation problem.

components act as service requesters to a collection of replicated service providers.

Although an analogous methodology has been studied previously in the context of internet congestion control, it has never been applied within the context of an Enterprise Services Bus or to request routing in a distributed web hosting environment. A major contribution of this work is the design of a decentralized market-based approach to resource allocation. This approach is a novel use of market-based control over shared resources within an ESB environment and a distributed web hosting environment. We also demonstrate through simulation that our decentralized market-based control mechanism is capable of providing near optimal resource management in an ESB setting.

In the next section, we present an example derivation of a decentralized routing mechanism for the distributed web hosting environment. Using the concepts developed in this example environment, Section 10.3 provides a more detailed view of the ESB system model

and develops an analogous decentralized approach. In Section 10.4, we analyze the robustness of this approach to resource allocation in the replicated ESB environment. Sections 10.5 and 10.5.2 present our simulation setup and results for the evaluation of this approach in a real-world setting.

10.2 Web Hosting Example

10.2.1 System Model

The environment in this example is that of a heterogeneous, distributed computing system designed to service a high-volume web site of world-wide interest. The system being modeled was used to implement a portion of the 1998 World Cup web site [8] that processed more than 1.3 billion HTTP requests during the summer of 1998. The web site was provided to a world-wide audience by four heterogeneous, geographically dispersed systems, each with their own processing capacity and workload distribution techniques. This class of system is very challenging to implement but occurs surprisingly frequently. The World Cup football tournament is just one example of an event of world-wide appeal that necessitates web-based coverage. Sites of this type are typically constructed for specific events, and the volume of traffic is on the order of billions of requests processed in a period of only a few months or less. Other such events that are reasonably expected to draw the attention of a world-wide audience in the billions might include the Olympics or the Tour de France.

Mapping requests to service providers is challenging because of the large volume of requests that must be processed. To help cope with this large volume of requests, the providers of the World Cup web site [8] chose a hierarchical deployment of the web site, where multiple instances of the distributed web hosting environment are deployed throughout the world. This work focuses on the application of our decentralized resource allocation mechanism to the design of a single instance of a distributed web hosting environment.

Figure 43 presents a graphical depiction of the distributed web hosting environment employed to deliver the 1998 World Cup web site. In this example system, users send requests for data to a web site over the Internet. These incoming requests are first routed to an instance of the web hosting environment by a network-layer load balancer (e.g., [40]).

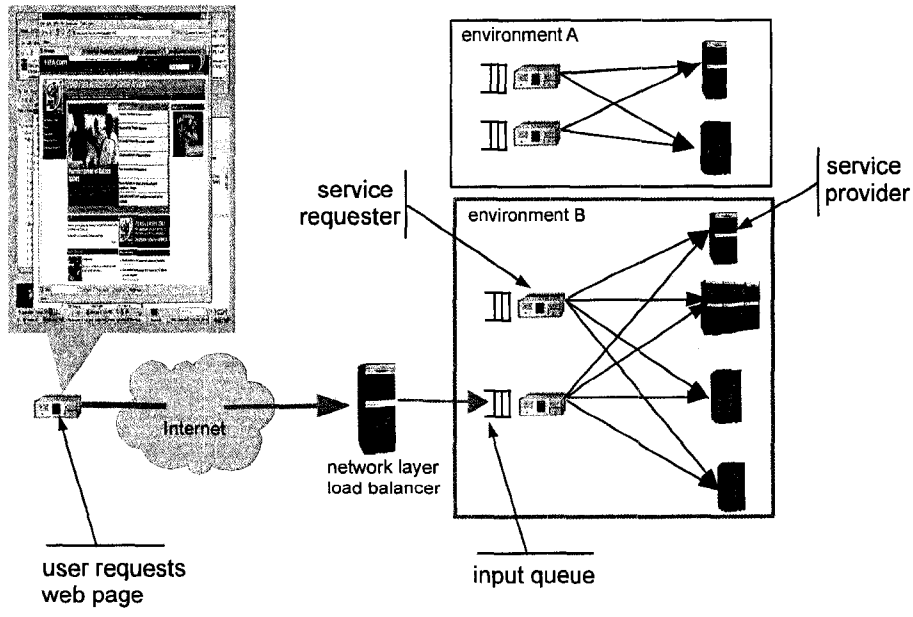


Figure 43: An example deployment of a distributed web-hosting environment. The system is designed to provide a web-site of world-wide appeal. Users send requests for data to a web site over the Internet. These incoming requests are first routed to an instance of the web hosting environment by a network layer load balancer (e.g., [40]). Incoming requests are queued for a service requester within the environment. The service requester applies our decentralized allocation mechanism to route each incoming request to one of the service providers. The service provider processes the incoming request and returns the results to the user.

A request in this environment is defined to be a menu-driven HTTP request for data that originates with a user. When incoming requests arrive to a web-hosting environment they are placed into the input queue of a service requester within that environment. Acting on the user's behalf, service requesters remove incoming requests from the input queues and route them to a service provider (i.e., web server) for processing. The service provider processes the request and returns the results to the user.

In our model, each distributed web hosting environment utilizes a collection of service requesters to route incoming requests to web servers for processing. Each web server is defined to be a service provider capable of servicing any incoming request.

Let $\mathcal{R}$ denote the set of all service requesters and let $\mathcal{P}$ denote the set of all service providers. Each service requester r ($r \in \mathcal{R}$) routes each incoming request to a service provider p ($p \in \mathcal{P}$). Service requester r produces a variable number of requests at each time

step k, denoted $f_r(k)$, based on the arrival rate of user driven requests. Each link connecting service requester r and service provider p is assumed to have a finite capacity for moving requests, denoted $c_{rp}^{(\text{in})}$. Finally, each service provider p can only process a limited number of requests in each time step, denoted c_p.

For each service requester, the fraction of its incoming requests that are sent to service provider p is denoted g_{rp}. In this example, each service requester r uniquely quantifies the value of using each service provider p with a simple scalar measure of value, denoted s_{rp}, e.g., speed. For a given service requester r and service provider p, we can define an aggregate service quality delivered by the system as:

$$\sum_p g_{rp} s_{rp}. \tag{64}$$

Intuitively, the service quality delivered by the system for traffic sent by service requester r can be thought of as the weighted average route quality realized for this traffic, where the g_{rp} values provide the weights. Finally, the system realizes some utility from delivering service requests to their destination. We account for differing service quality by quantifying utility, denoted $U(x)$, as the worth of receiving service quality x. In our model, we have chosen a utility function that depends on a scalar quantification of service quality. However, our technique is applicable to a more general utility function that accepts the full vector of g_{rp} and s_{rp} values to compute a scalar utility value.

10.2.2 Centralized Optimization

We initially model resource allocation in this system as a constrained optimization problem that will serve as the basis for the derivation of our decentralized approach to resource allocation. We combine the elements of our system model presented in the previous subsection to form the following optimization problem:

$$\text{maximize} \quad \sum_r f_r(k) U\left(\sum_p g_{rp} s_{rp}\right) \tag{65}$$

$$\text{subject to:} \quad \forall\, r, \sum_{p} g_{rp} = 1 \tag{66}$$

$$\forall\, r,p, \quad g_{rp} \geq 0 \tag{67}$$

$$\forall\, r, \quad g_{rp} f_r(k) \leq c_{rp}^{(\text{in})} \tag{68}$$

$$\forall\, p, \sum_{r} g_{rp} f_r(k) \leq c_p \tag{69}$$

Intuitively, for each service-requester–service-provider pair Equation 65 sums the realized utility of the service quality scaled by the production rate for that service requester. By taking this sum over all service requester-provider pairs we are evaluating the total realized utility for the system at time step k.

The four constraints that must be satisfied for this optimization enforce the capacity limitations of the system. First, Equation 66 requires that, for each service requester r, all of the requests received in a given time step are sent to a service provider for processing, and Equation 67 enforces the fact that negative decision variables are meaningless in this context. Equation 68 enforces the constraint on processing capacity for each service provider. Equation 69 enforces the constraint on capacity for each service provider.

The solution to this constrained optimization problem provides an optimal allocation of system resources at any given time step k. The solution can be applied by a centralized resource manager to allocate requests to service providers. However, because the production rates of requests change from one time step to the next, the optimal allocation of requests also changes from one time step to the next. In a problem of reasonable size, the time required to solve the constrained optimization problem may be longer than a single time step. That is, because the optimal allocation changes with time, a centralized approach may continually lag behind the optimal allocation.

10.2.3 Decentralized Approach

To transform the original optimization problem described in the previous subsection into a decentralized algorithm for resource allocation, we apply the Lagrangian multiplier method [19,21,30]. In this way, we convert the centralized optimization problem into an equivalent

problem that can be separated into $|\mathcal{R}|$ independent sub-problems where each of the sub-problems is much simpler to solve. The constraint of Equation (69) in the centralized problem reflects a constraint on *shared* resources. The Lagrangian method provides a mechanism for introducing Lagrange multipliers (interpreted as "prices") for these shared resources that can be directly accounted for during optimization as a penalty. An important feature of this approach is that solving for the g_{rp} values for each service requester no longer requires detailed knowledge of the g_{rp} values of the other service requesters. These detailed values are instead replaced by a collection of prices produced by the service providers, where each price reflects the current demand for the capacity of that service provider. Let $\underline{\phi_p(k)}$ denote the price of using service provider p ($\forall p \in \mathcal{P}$) at time step k. Each service requester r must choose g_{rp} values to solve the following problem:

$$\text{maximize} \quad f_r(k) \sum_p \left[U\left(g_{rp} s_{rp}\right) - \phi_p(k) g_{rp}\right] \tag{70}$$

$$\text{subject to:} \quad \sum_p g_{rp} = 1 \tag{71}$$

$$\forall\, p, \quad g_{rp} \geq 0 \tag{72}$$

$$g_{rp} f_r(k) \leq c_{rp}^{(\text{in})} \tag{73}$$

The above optimization problem only depends on the prices obtained from each service provider and information that is locally available to the service requester. The maximization of this simpler problem is modified to account for the cost of using each service provider. In this way, the constraint on service provider capacity is accounted for as a penalty as opposed to a direct constraint [30]. However, the remaining constraints on local resources must still be enforced as before.

By reformulating the centralized optimization problem in this manner, we are able to obtain a collection of sub-problems whose combined solution is mathematically equivalent to the solution of the original centralized optimization problem. The final required ingredient

in this model is an algorithm for computing the appropriate price for the consumption of capacity at each service provider.

Each service provider p is required to update the price in the model for its capacity such that the price reflects expected near-term future demand for the resource. The price of each shared resource reflects the amount of excess capacity that the resource has for processing requests. If the demand for a shared resource is greater (less) than the supply, then the price of the resource should increase (decrease). The price of each shared resource is updated according to the difference between the capacity of the shared resource and current demand for the resource scaled by a constant factor. In our model, we introduce a constant step-size parameter for controlling price updates for service provider capacity, denoted $\underline{\nu}$, whose value must be greater than 0. Each service provider p can simply update prices $\phi_p(k)$ using the following update procedure where $[x]_+ = \max\{x, 0\}$:

$$\phi_p(k+1) = \left[\phi_p(k) - \nu\left(c_p - \sum_r g_{rp} f_r(k)\right)\right]_+ .$$

The update procedure is intentionally expressed in terms of abstract time-steps. Each pair of successive time-steps can be mapped to a specific interval of real time. There are a variety of possible interpretations of the real time that passes during the interval between any pair of successive time-steps k and $k+1$. One natural interpretation treats all of the time intervals as having the same constant length, i.e., each interval corresponds to the same amount of real time.

The step-size ν determines how the system will react to fluctuations in demand for shared resources. Notice that the step-size determines the magnitude of price updates. That is, if ν is too small, then the prices will be slow to react to changes in demand. For example, if price updates are too small and the demand for a resource is much greater than its capacity, then the price for the shared resource may not increase enough to deter future requesters from using it until a much later time. Consequently, the number of queued requests in the input buffer of the shared resource may increase because it is consistently receiving more requests than it can process. The value ν needs to be large enough to enable the system

to react to substantial changes in demand, i.e., increase the price enough to deter recurrent excessive demand. Care must also be taken to prevent the opposite scenario, where prices are increased so much that future demand for the shared resource is unnecessarily reduced to 0. That is, if ν is set too high, then the system may thrash, i.e., demand may oscillate between shared resources potentially overwhelming some resource in any given time step.

An example to illustrate communicating prices is shown in Figure 44. In this example, service requester r_1 is sending requests to service providers p_1 and p_2. To facilitate the allocation decisions made by r_1, each of the service providers provide prices for using each of the shared resources that it is responsible for in the system, e.g., the capacity of service provider p_1.

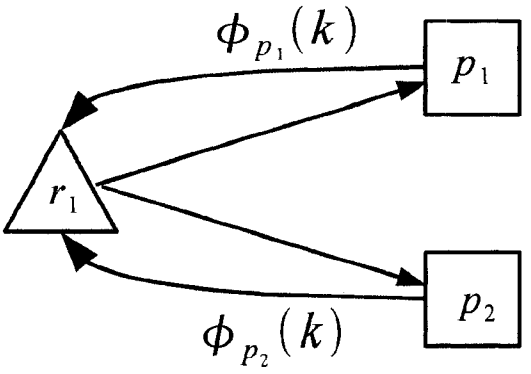


Figure 44: An example of the price update procedure. In the example, service requester r_1 receives updates for the price of using service provider p_1 and service provider p_2.

10.3 Replicated ESB

10.3.1 System Model

In this section, we introduce a more complex system model involving multiple shared resources. This example system is based on modeling a replicated ESB as an overlay network (shown in Figure 41). In this computing system, there are three types of entities: service requesters, ESB components, and service providers. As in our previous example, service requesters send requests to service providers, however, in this example, requests are routed to the service providers by a collection of independent ESB components. In our previous

example, the service providers all performed the same function, i.e., they all served up pages from the same web-site. In this example, each service requester is assumed to provide a unique service to the system. Service requesters request a service by name and it is the responsibility of the ESB components to route incoming requests to a service provider capable of processing the request. It is helpful to consider a service requester in this context as an agent that is making requests to an ESB on behalf of an application that is external to the replicated ESB system. That is, an external application passes requests to the service requester which makes routing decisions to ESB components on behalf of the application. In this way, the service requester can be treated as a component of the overall ESB infrastructure instead of as an outside entity.

Let $\underline{\mathcal{E}}$ denote the set of all ESB components, let $\mathcal{R}$ continue to denote the set of all service requesters, and let $\mathcal{P}$ continue to denote of the set of all service providers. In this system, rates of requests from individual service requesters are not required to be observable; instead, the system only requires that each ESB component be able to measure the number of requests that have arrived in each time step—a value readily available by inspection of the input buffer of each ESB component. In this system, each service provider is assumed to provide a unique service to the system. Thus, each service requester r ($r \in \mathcal{R}$) produces requests for a service provider p ($p \in \mathcal{P}$) at a rate of $\underline{f_{rp}(k)}$ requests per time step k. Service requesters are modeled as having an output buffer, and all requests produced by the requester are written to this output buffer prior to transmission. Requests are transmitted from the output buffer of the service requester using the appropriate overlay network link to the input buffer of the selected ESB component. Each ESB component processes requests from its input buffer, forwarding each request to its chosen service provider.

In the example of Figure 41, service requesters are connected to all of the ESB components, i.e., there is a finite-capacity link connecting each service requester to each ESB component, where the finite-capacity of each link is modeled as a constraint on the transmission rate through that link. Thus, a link between a service requester $r \in \mathcal{R}$ and an ESB component $e \in \mathcal{E}$ is subject to a capacity constraint $\underline{c_{re}^{(\text{in})}}$ such that the rate of requests sent from service requester r to ESB component e cannot exceed $c_{re}^{(\text{in})}$ in any given time step.

Requests received by an ESB component are assumed to be buffered in a finite-capacity input buffer for that ESB component. Each ESB component also maintains a finite-capacity output buffer for storing requests that have been processed and are ready to be transmitted to an appropriate service provider. Each ESB component $e \in \mathcal{E}$ is subject to its own capacity constraint on its computing capabilities such that the rate at which each ESB component e can process requests is limited to $\underline{c_e}$. That is, an ESB component e with processing capacity c_e can move at most c_e requests from its input buffer to its output buffer in a single time unit.

The outgoing links from each ESB component to its attached service providers are subject to capacity constraints. The link connecting an ESB component e to a service provider $p \in \mathcal{P}$ has a finite capacity $\underline{c_{ep}^{(\text{out})}}$ to move data from the output buffer of ESB component e to the input buffer of service provider p. Lastly, in this example, there are no modeled constraints on the capacity of the service providers. That is, for this model we chose to focus on applying the market-based resource allocation strategy to the ESB components. However, it should be clear based on our earlier presentation of the web hosting environment that the presented approach could easily be extended to include allocation decisions regarding multiple service providers for the same class of service. That is, by applying the model of Section 10.2 to the selection of service provider replicas, the capacity of each service can be extended to accommodate excessive demand.

There are two types of shared resources: the ESB components and the links connecting ESB components to the service providers. For these shared resources, there is a difference between the advertised capacity (c_e and $c_{ep}^{(\text{out})}$) of a shared resource and the physical capacity of the underlying resource. The physical capacity of any given shared resource can be viewed as the "true" capacity afforded by the physical limitations of the device, i.e., the physical capacity is an upper bound on the best possible performance of the device. If a system were sized such that it were required to operate at its physical limit for the duration of its execution, then if that system were ever asked to process more than that limit, it would be incapable. It is incumbent upon each ESB component e to construct prices from its advertised capacity and not from true capacity. Specifically, the advertised capacity is given

as some reasonable fraction of the true system capacity, e.g., we can define the advertised capacity as ρ times the true capacity where $\rho \in (0, 1)$ is often called the "load factor." By communicating availability in terms of advertised capacity, the ESB component is insulated from instantaneous fluctuations that occur during the normal course of system operation that might otherwise overwhelm the component. Thus the capacity for each shared resource used in our model (c_e and $c_{ep}^{(\text{out})}$) is assumed to be the advertised capacity.

The Information Technology industry is attempting to recover the cost of maintaining Wide Area Network (WAN) links by billing for network traffic that is transported across these links. That is, some companies are beginning to monitor WAN traffic volume in an attempt to bill customers for the amount of data transferred across these expensive network links. In a globally distributed system such as a replicated ESB, network link costs need to be accounted for during resource allocation. By introducing a pricing scheme for WAN links, the network provider has created a situation where some links are more valuable than others. To help illuminate the impact of such a decision, consider the following simple example. Using the model of Figure 41, assume that one ESB component is physically located in Fort Collins, CO, in the U.S.A., and another is located in Bangalore, India. If a service requester located in the U.S.A. intends to communicate with a service provider also in the U.S.A., then the lowest cost route for this traffic may be through the ESB component located in Fort Collins. In this example, by sending traffic through the U.S.A. based ESB component, a network charge can be avoided. Although somewhat exaggerated, this example illustrates the heterogeneity of the available routes. The system model accounts for this heterogeneity by incorporating a quantification of route quality into the optimization problem. The quality of each route from service requester r through ESB component e to a service provider destination p is given a single numeric value, denoted s_{rep}, quantifying the quality (e.g., speed) of the route from the perspective of each service requester.

The goal of this model is to help ascertain an optimal allocation of ESB components and associated links to service providers for transmitting service requests. Recall, each service requester r produces $f_{rp}(k)$ service requests per time step k for a particular service provider p. This traffic is sent through some ESB component in $\mathcal{E}$ and all of the traffic must be

eventually transmitted to its destination, e.g., to some service provider p. Thus, we can identify the percentage of requests from a service requester r sent to a service provider p that are transmitted through ESB component e, denoted g_{rep}. The goal of a resource management heuristic in this environment is to choose a combination of the g_{rep} values such that a system-wide goal is maximized.

In this example, we assume that a service requester receives some utility from the successful transfer of a request to a service provider. For example, the service provider may provide a printer repair service—the data being transferred by the service requester might be a notification of a printer outage. Delivering this service request to an appropriate service provider enables the provider to dispatch a technician to repair the broken printer. That is, each service requester can realize some quantifiable utility from each request that is successfully delivered through the system. Additionally, some service requesters may be considered more important than others. For example, the system provider may wish to provide a higher level of service to some special customers than what is normally offered. To model this behavior, we prioritize the traffic sent by each service requester to each service provider. Let θ_{rp} denote the priority of requests sent from service requester r ($r \in \mathcal{R}$) to service provider p ($p \in \mathcal{P}$).

For a given service requester r and service provider p, we can define an aggregate service quality delivered by the system as:

$$\sum_{e} g_{rep} s_{rep}. \tag{74}$$

Intuitively, the service quality delivered by the system for traffic sent by service requester r to service provider p can be thought of as the weighted average route quality for this traffic, where the g_{rep} values provide the weights. As described earlier, the system realizes some utility from delivering service requests to their destination. As in our previous example, we account for varying service quality in the model by quantifying utility, denoted $U(x)$.

10.3.2 Centralized Optimization

In a manner similar to our previous example, we pose the replicated ESB resource allocation problem as an optimization problem. The resulting optimization can be directly solved using

a centralized approach by a system-wide resource manager. Using the definitions and system constraints from the previous section, the following constrained optimization problem can be defined, where each g_{rep} value is chosen such that the following objective is maximized:

$$\text{maximize} \sum_{r,p} \theta_{rp} f_{rp}(k) U \left(\sum_{e} g_{rep} s_{rep} \right) \tag{75}$$

$$\text{subject to:} \quad \forall\, r,p, \sum_{e} g_{rep} = 1 \tag{76}$$

$$\forall\, r,e,p, \quad g_{rep} \geq 0 \tag{77}$$

$$\forall\, r,e, \sum_{p} g_{rep} f_{rp}(k) \leq c_{re}^{(\text{in})} \tag{78}$$

$$\forall\, p,e, \sum_{r} g_{rep} f_{rp}(k) \leq c_{ep}^{(\text{out})} \tag{79}$$

$$\forall\, e, \sum_{r,p} g_{rep} f_{rp}(k) \leq c_{e} \tag{80}$$

In the proposed centralized problem, the realized utility of the overall achieved service quality is scaled by the priority and volume of the traffic. Equation (75) expresses the overall goal of the optimization to maximize the realized utility given a collection of service requesters each with their own priority. The constraint of Equation (76) ensures that all of the requests from a given service requester are routed through the overlay network. Equation (77) ensures that the g_{rep} values are positive and Equations (78), (79), and (80) enforce the capacity constraints for each of the constrained system resources.

This centralized approach could be a reasonable solution for the special case of a static rate of requests from each service requester. That is, if the rate of requests produced by all requesters in the system remains constant, then it may be reasonable to calculate a solution to the centralized problem off-line. However, if the rate of requests to be processed is changing with time, then the centralized solution becomes infeasible to maintain for all systems of reasonable size because the optimal solution to the problem changes faster than the centralized solution can be calculated. Similarly, if the centralized problem requires too

many decision variables, i.e., there are too many service requesters, service providers, or ESB components, then it may be unreasonable to calculate the solution in advance off-line. That is, the centralized approach to resource allocation does not scale well.

In this environment, because the rate of requests sent from each service requester to each service provider, i.e., $f_{rp}(k)$, is a function of time, this centralized form of the problem must be solved whenever any of the request production rates change. However, for any problem of reasonable size, the computation time required to solve this problem makes it difficult to complete the solution prior to the request production rates changing again. In the next section, we identify an equivalent solution that is much faster to calculate and does not require central control.

10.3.3 Decentralized Approach

To transform the centralized optimization problem into a decentralized algorithm, we again apply the Lagrangian multiplier method to the centralized optimization problem to define an equivalent problem that is separable into $|\mathcal{R}|$ independent sub-problems. The constraints of Equations (78) and (79) in the centralized problem reflect constraints on *shared* resources. An important feature of this approach is that solving for the g_{rep} values for each service requester no longer requires detailed knowledge of the g_{rep} values of the other service requesters. These detailed values are instead replaced by a collection of price vectors that are produced by the ESB components, where each price vector reflects the current demand for shared resources attached to that ESB component. Let $\underline{\pi_e(k)}$ denote the price of ESB component e ($\forall e \in \mathcal{E}$) at time k and let $\underline{q_{ep}(k)}$ be the price for a link from component e to service provider p ($\forall e \in \mathcal{E}$, $\forall p \in \mathcal{P}$) at time step k. Each service requester r must choose g_{rep} values to solve the following problem:

$$\text{maximize} \sum_{p} \theta_{rp} f_{rp}(k) U\left(\sum_{e} g_{rep} s_{rep}\right) - \sum_{e,p} \left(\pi_e(k) + q_{ep}(k)\right) g_{rep} f_{rp}(k). \qquad (81)$$

Service requester r also must enforce the following constraints corresponding to Equations (76), (77), and (78) from the centralized problem,

$$\forall\ p, \qquad \sum_{e} g_{rep} = 1 \tag{82}$$

$$\forall\ e,p, \qquad g_{rep} \geq 0 \tag{83}$$

$$\forall\ e, \sum_{p} g_{rep} f_{rp}(k) \leq c_{re}^{(\text{in})}. \tag{84}$$

The prices $\pi_e(k)$ and $q_{ep}(k)$ $(\forall e \in \mathcal{E}, \forall p \in \mathcal{P})$ are obtained from each ESB component e as input to the above model in the form of an update $\pi_e(k) + q_{ep}(k)$ $(\forall p \in \mathcal{P})$ received at every time step k.

To complete the decentralized algorithm, we require a mechanism for computing the prices for each of the ESB components and the links connecting them to the service providers. Each ESB component e is required to update prices in the model for its shared resources such that the prices reflect expected near-term future demand. The price of each shared resource reflects the amount of excess capacity that the resource has for processing requests. In this model, we differentiate between updates to ESB component prices and link prices by introducing two different constant step-size values, one for the ESB component prices, denoted $\underline{\alpha}$, and one for the link prices, denoted $\underline{\gamma}$—both step-sizes must be greater than 0. The ESB component e can simply update prices π_e and q_{ep} $(\forall p)$ using the following update procedure:

$$\pi_e(k+1) = \left[\pi_e(k) - \alpha\left(c_e - \sum_{rp} g_{rep} f_{rp}(k)\right)\right]_+$$

$$q_{ep}(k+1) = \left[q_{ep}(k) - \gamma\left(c_{ep}^{(out)} - \sum_{r} g_{rep} f_{rp}(k)\right)\right]_+$$

As in the previous example, care must be taken in selecting an appropriate step-size to ensure that prices react expeditiously to market fluctuations.

10.3.4 Resource Management Using this Approach

Each service requester r must have a procedure for utilizing the results obtained by solving the local optimization problem for the g_{rep} values. Because the service requester cannot send fractions of a record, we need to approximate the direct use of the g_{rep} decision variables. The simplest mechanism for converting the g_{rep} decision variables into a resource allocation is to interpret their values as probabilities. Recall that the g_{rep} values are constrained to the interval [0,1]. Thus, each g_{rep} value can be interpreted as the probability that any given record will utilize the route from service requester r through ESB component e to service provider p. We can apply the g_{rep} values to the route selection process, by constructing a cumulative distribution function (cdf) where each g_{rep} value is interpreted as the probability of selecting its associated route. The service requester then indexes into the constructed cdf using a generated uniform random number in the range [0,1]. Using the g_{rep} values in this way will, over many sent records, approximate the direct use of the g_{rep} values.

10.4 Robustness of the Decentralized Approach

In a real-world computing environment, network traffic is often triggered by world events outside the control of the computing system. For example, a financial market sell off could reasonably be expected to result in an increase in network traffic communicating market sell orders to the financial market. In a similar manner, changes in the volume of service requests that a replicated ESB must process is a source of system uncertainty. That is, a resource manager responsible for allocating ESB components to service requests cannot accurately predict the upcoming volume of requests that will need to be serviced.

A system can be considered robust to perturbations in system parameters, if the change in system performance due to this uncertainty is limited [3]. In this system, we utilize an overall performance measure that accounts for requester priorities, the number of requests being transferred, and the quality of the network routes being used. Requests are produced by service requesters at some rate and transmitted to ESB components where they are buffered before being forwarded on to their final destination. Because the system is decentralized and the production rate of service requests is changing with time, it is possible

for the system to experience contention for shared system resources. When contention for shared resources occurs, potentially due to changes in the production rate of requests, it is possible for low-priority requests to inhibit the transfer of high-priority requests causing the performance measure to degrade.

Intuitively, the robustness [3, 25, 89, 93] of the decentralized allocation approach can be established by answering the following three questions. What behavior of the system makes it robust? What uncertainties is the system robust against? Quantitatively, exactly how robust is the system? Uncertainty in service request production rates can directly impact the system performance measure. In this system, we might consider a resource allocation strategy robust as long as it maintains a performance measure that is within $X\%$ of the optimal value, where X is a user defined constraint on the acceptable performance of the system. We can quantify robustness in this environment as the proximity of the system performance measure to its optimal value. Based on this intuitive understanding of robustness in this context, we can utilize the FePIA procedure [3] to derive a more formal definition of robustness.

In step 1 of the FePIA procedure, we derive a robustness requirement that clearly defines robust operation of the system. Let the realized utility of the system be defined as:

$$\Psi(k) = \sum_{r,p} \theta_{rp} f_{rp}(k) U \left(\sum_{e} g_{rep} s_{rep} \right). \tag{85}$$

From our intuitive definition of robustness, we can state that the system is robust if at each time step k the realized utility $\Psi(k)$ remains within an acceptable range. Let $\beta_{\min}(k)$ be the smallest acceptable realized utility value where $\beta_{\min}(k)$ is expressed in terms of the optimal realized utility value as given by the centralized solution found at time step k. That is, given an optimal utility value measured at time step k, denoted $\omega(k)$, let $\beta_{\min}(k) = X\omega(k)$ where X is in the range (0,1). The robustness requirement for our system can be expressed as $\beta_{\min}(k) \leq \Psi(k)$. That is, if the realized utility at each time step k remains larger than $\beta_{\min}(k)$, then the system can be considered robust.

The principal source of uncertainty in this system that may cause fluctuation in $\Psi(k)$

is the variability in the production rate of requests. Recall that the production rate of requests $f_{rp}(k)$ is not known in advance and may differ from one time step to the next. This variability in the production rate may directly impact the realized utility and consequently $\Psi(k)$. Finally, the robustness of a resource allocation can be quantified as the smallest $\Psi(k)$ value that occurs over all time steps k. That is, the robustness of a resource allocation, denoted $\underline{\Psi}$, can be measured through time step k and expressed as:

$$\Psi = \min_{k} \Psi(k). \tag{86}$$

10.5 Simulation Study

10.5.1 Setup

Several simulations were conducted to evaluate both the accuracy of the implemented system as well as the efficacy of the overall approach. We evaluate the approach in a realistic application of the replicated ESB system where the production rate for each service requester varies as a function of time. The variable production rate of records in the system is modeled using a simple scaled sinusoid that provides a periodic change in record production unique to each service requester. The superposition of the record production functions of the service requesters is presented in Figure 45(a). Each point in the plot corresponds to the number of records produced in the simulation at that time step and presents a view of the load placed on the overall system in that time step. The combination of sinusoids chosen is such that the summed traffic is periodic, repeating the exact same pattern of production approximately every 2200 time steps.

The plot of Figure 45(a) can be scaled according to the priority of each service requester service provider pair (θ_{rp}) producing a plot (Figure 45(b)) of the optimal utility given homogeneous network routes, i.e., assuming all routes are of equal value ($s_{rep} = 1 \quad \forall r, e, p$). Ideally, the solution found by the described decentralized routing mechanism will track this optimal time-varying utility. The simulations conducted consisted of four service requesters, two ESB components, and three service providers, where the collective production rate of the service requesters is varied according to Figure 45(a).

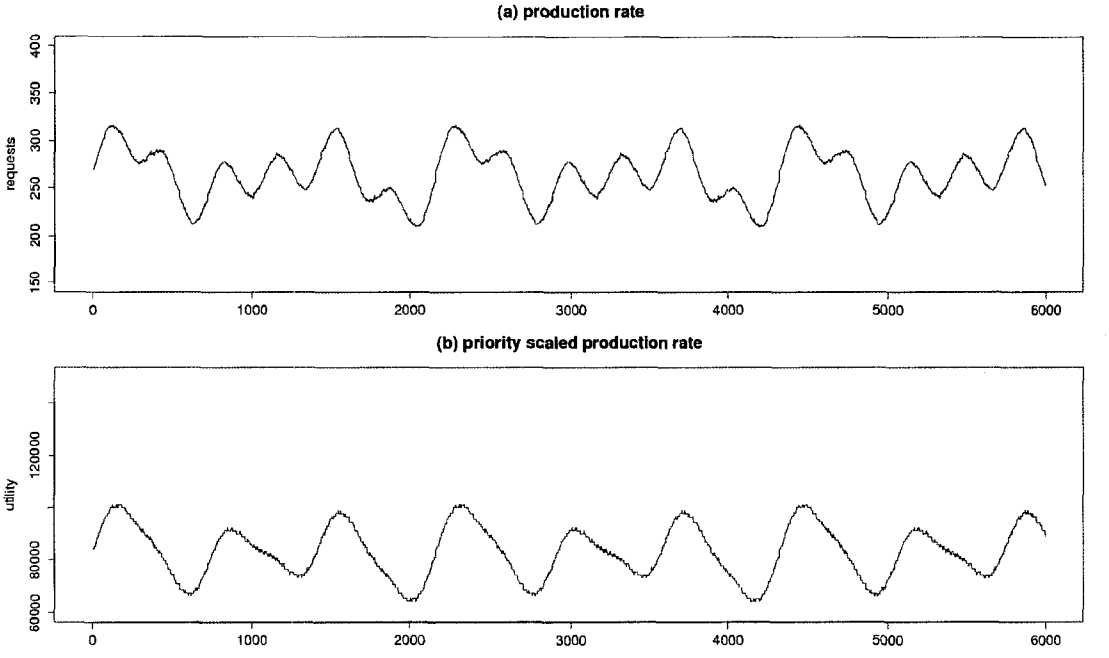


Figure 45: (a) The summed production rates over all service requesters plotted versus simulation time step; (b) the summed production rates scaled by service requester priority plotted versus simulation time step.

10.5.2 Results

For this simulation, we compared the results of the centralized solution to the results of our decentralized solution to demonstrate the effectiveness of the approach. The comparison was made by periodically extracting the information required to produce the centralized optimization problem from the details of the dynamic simulation at a given time step. We solved these instances of the centralized problem offline and compared the centralized result to the result produced by our decentralized approach. Recall that the centralized solution provides an optimal solution as described in Section 10.3.

In a real-world environment, the rate of requests submitted to the system will vary with time in an unpredictable manner. To assess the viability of our approach, we modeled the production rate of requests from service requesters as a function of time. Thus, at each instant in time the optimal allocation is given by a unique optimization problem with its own unique solution. Consequently, a centralized solution in this environment is

impractical because it would require recalculating the entire solution at every time-step. In the decentralized approach, each service requester can effectively solve their own optimization problem locally using the prices provided by the ESB components along with information that is locally available to each requester. The combined results of all of the service requesters should be equivalent to that found using the centralized solution—the decentralized implementation should be capable of tracking the optimal solution as it varies over time.

Figure 46 presents the results for a sample implementation where service requester production rates are functions of time and all routes in the system are of equal quality, i.e., $s_{rep} = 1 \quad \forall r, e, p$. The three plots of Figure 46 present (a) the total number of requests at each time step during the simulation, (b) the realized utility over time, and (c) the average price for shared resources in the system over time.

Embedded within the plot of realized utility (Figure 46(b)), we have plotted the exact centralized solution overlaid as circles on top of the decentralized solution. From the plots of the figure, we can clearly see that the decentralized solution tracks the periodic results of the centralized solution. However, there is a slight (less than 1%) fluctuation in performance due to the buffering of data within the system. That is, each service-requester – service-provider pair has a different priority and in any given instant there may be minor contention for a shared resource (i.e., an ESB component or ESB to service provider link). When this contention occurs, some high-priority traffic may be delayed due to the system processing lower priority traffic first. The realized utility will be temporarily reduced as a result of the delay but will increase in some subsequent time step as the delayed records arrive at their destination. Thus, the realized utility from the decentralized solution in this later time-step will appear to be greater than optimal. Because of contention and request delays at a previous time-step, the decentralized and centralized approaches are considering different sets of requests.

An alternative approach to evaluating the results of the decentralized solution is possible in this case, i.e., when the quality of the given routes are homogeneous ()$s_{rep} = 1 \quad \forall r, e, p$). In this case, it is possible to compare the results of the decentralized solution to the sum

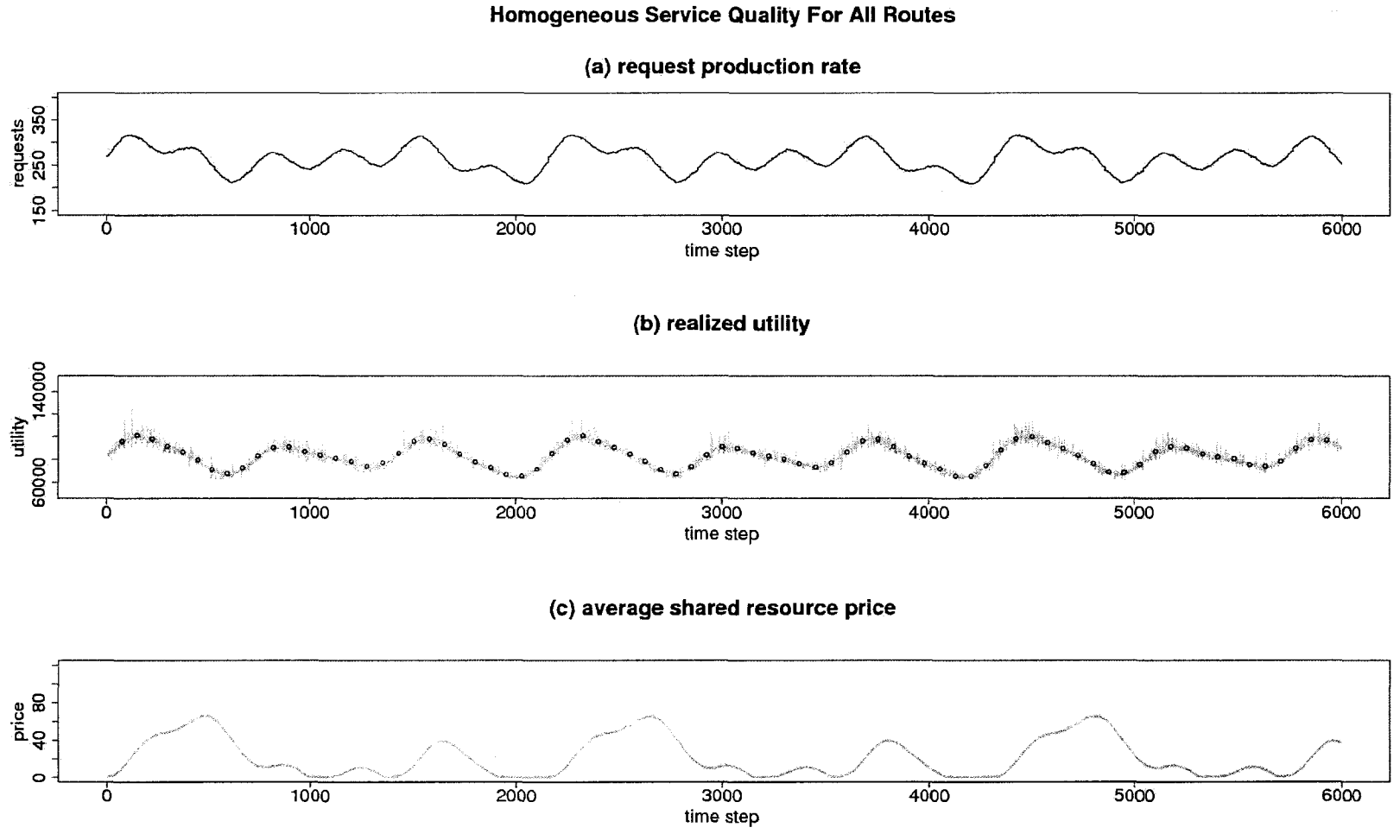


Figure 46: Sample results for a homogeneous network, i.e., $s_{rep} = 1 \; \forall r, e, p$, given service request production rates that vary as functions of time. Plot (a) of the figure presents the collective request production rate for the system as a function of time. Plot (b) presents the realized utility for our decentralized approach (plotted as a line). Periodically, throughout the simulation, an equivalent centralized optimization problem was extracted from the state of the simulation at a given time-step and solved. These results are plotted as circles in Plot (b) overlaid on top of the decentralized solution. Plot (c) presents the average shared resource price in the network as a function of time.

of the scaled request production rate over all service requesters. In other words, we sum the decentralized realized utility over each time step for the entire simulation and do the same for the request production rate scaled by priority. Ideally, these two values should be identical given homogeneous route quality. Our decentralized technique realizes 99.5% of the scaled production rate result.

Our second simulation includes routes of differing quality, where the s_{rep} values were different for each route. For this simulation, individual s_{rep} values were selected at random in the range [1,10]. Recall that the s_{rep} values appear in the utility function as a multiplier for the g_{rep} values. In this way, the g_{rep} values are scaled according to the s_{rep} values prior to calculating utility. Consequently, a centralized controller might attempt to maximize the likelihood that high priority traffic will be assigned to its best route, thus maximizing the realized utility. In Figure 47, we can see that given heterogeneous routes the results of the decentralized solution still effectively track the results of the centralized solution.

In both Figures 46(c) and 47(c), we plotted the average shared resource price for the entire simulation. Notice that the prices are periodic with a period identical to that of the production rates plotted in Figures 46(a) and 47(a), respectively. For these simulations the system appears stable. That is, because the request production rate is periodic, a stable system would imply that the average shared resource price would return to its starting point at the end of any given period. Recall that the periodicity of the request production rate is such that the pattern repeats approximately every 2200 time steps; similarly the average shared resource price is periodic repeating the exact same pattern every 2200 time steps. In our simulations, the initial configuration of the system represents a slightly over-provisioned system. Thus, the initial prices of the shared resources are all 0. At the end of the period, we should expect the average shared resource price to return to 0, and indeed, it does.

Our simulation study results clearly demonstrate the viability of our decentralized market-based approach to resource management in a heterogeneous distributed computing system such as the distributed ESB environment. The plots of realized-utility in both Figure 46(b) and Figure 47(b) demonstrate the approaches ability to successfully track the highest performance possible for such a system.

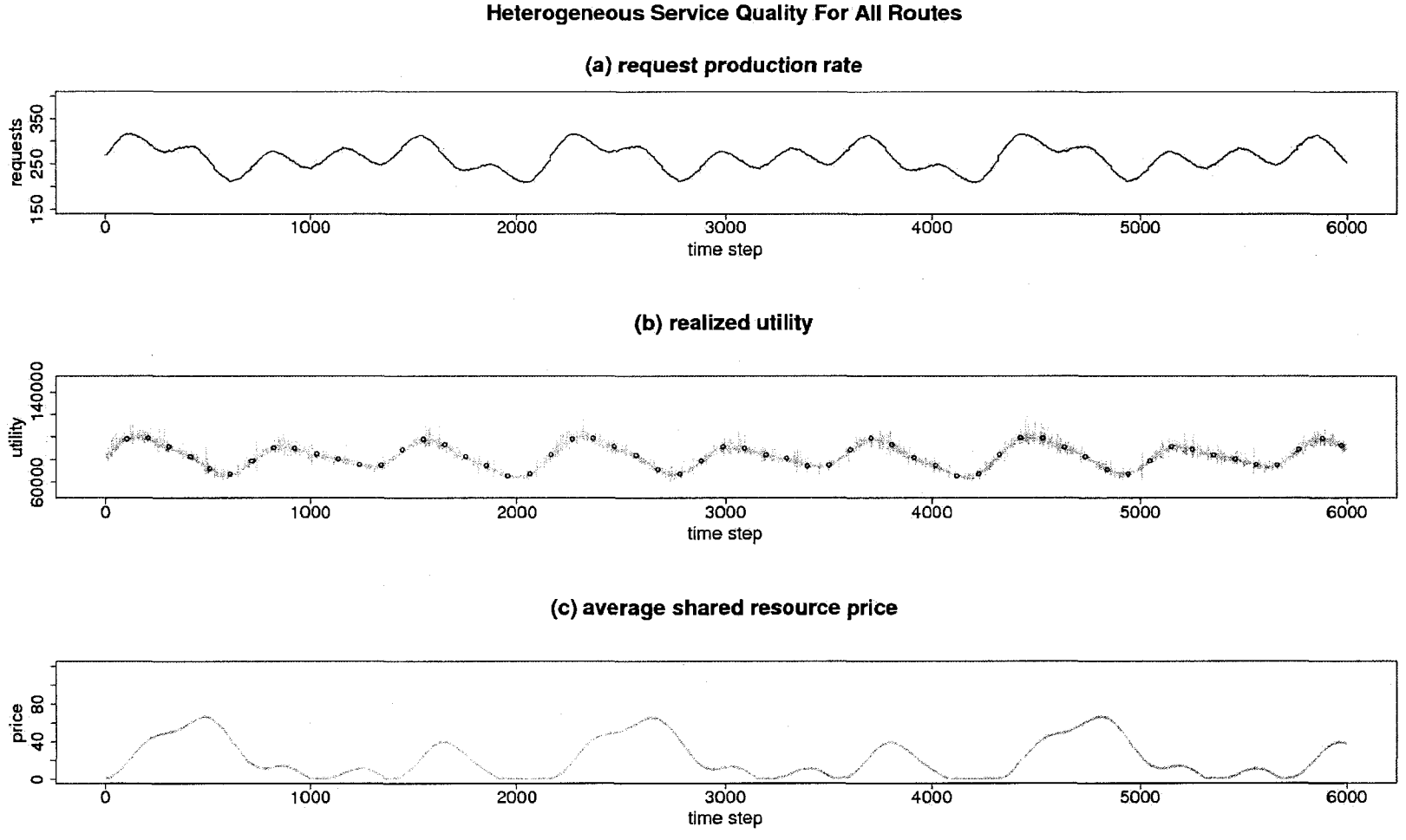


Figure 47: Sample results for a heterogeneous network, i.e., each route through the network connecting a service provider to a service requester has a unique service quality associated with it. Plots a, b, and c correspond to the same plots in Figure 46 for the homogeneous case.

10.6 Related Work

A related field to the study of an Enterprise Services Bus is that of a Content Delivery Network (CDN) commonly used to improve the apparent performance and reliability of web sites by distributing their content throughout the world wide web. In a CDN, web-site content is cached at replica servers that are capable of replying to web requests on behalf of the owning web site. In [79], the authors introduce the concept of a collaborative CDN (CCDN). A CCDN is described as being an overlay network that utilizes end-user machines in a peer-to-peer fashion to provide a CDN across a wide-area network. In the Globule system, user requests for data available on the CCDN are delivered to replica servers using a redirection service capable of HTTP redirection. In [92], the authors present the AS-path length heuristic used in Globule to provide a redirection policy for user requests. The AS-path length heuristic greedily redirects user requests to the closest replica server available in the CDN, where proximity is defined in terms of the number of network hops between the requester and the replica. This simple greedy approach does not account for contention among the shared resources of the CCDN, i.e., the replica servers. Because the Globule system is an instance of an overlay network it can be modeled as a trans-shipment network flow problem. By modeling an overlay network in this manner, our market-based resource allocation technique can be applied to the routing of web-site requests to replica servers based on current network load, where the proximity of the requester to the caches and the network bandwidth of the caches can be used to construct a quality value for each route in the CDN, i.e., s_{rep}. In this way, our approach can account for both the proximity of replicas to users as well as any contention for the shared resources of the CCDN, where contention is not considered in [92].

Another approach to market-based resource allocation involves the use of an auction as opposed to price setting. The Tycoon resource allocation system presented in [60] provides such an auction based market for resource allocation in a distributed system. In the Tycoon system, users bid for the right to use compute resources within the Tycoon network. Bids are accepted by a collection of auctioneers that manage access to Tycoon compute resources. In an auction system, the auctioneer must accept bids for a resource for some period of time

before closing the auction. The waiting period for an auction to close is acceptable as long as the time required to complete the auction is less than that of the task to be executed. In our environment, tasks are extremely short lived, e.g., the time required to produce a static web page or deliver a message in an overlay network. Consequently, the time delays incurred by an auction for a resource are infeasible within our context. However, prices for shared resources are set within our network based on current demand. Thus, our price setting approach is more analogous to a bid-ask auction system where the resource seller sets an asking price and the buyer accepts that price by purchasing the right to that resource. In this way, as demand for a shared resource fluctuates, so do the prices for that resource.

In [55], a system called WebSeAl is introduced that provides resource allocation in a CDN. One of the many claims of the WebSeAl system is its ability to balance the request load for a web site across multiple geographically dispersed replica web servers. Their approach to resource management of the server pool is to introduce prices for the use of servers in their network that force the clients to route their requests based on this price information. Clients in the WebSeAl environment make routing decisions based on a combination of performance data about the response time of each replica server and a weighting factor for each replica. The authors assume that clients in the WebSeAl environment will be "sensitive" to the weighting factor and account for current system weights while making resource allocation decisions. As the authors of [55] state, the WebSeAl environment is therefore best suited to serving web sites where there is no incentive to circumvent the balancing aspects of the system, e.g., web sites delivered on a corporate intranet. By ignoring the weighting factor a client may instead request solely based on selfish performance data, i.e., always selecting the replica that provides the best possible performance to the client.

Like the WebSeAl environment our system utilizes a price setting scheme to enable clients in the system to make routing decisions. However, unlike WebSeAl, prices in our environment are set based on direct feedback from the system regarding current demand for shared resources. Further, the clients (service requesters) in our system solve an optimization problem locally that leverages current prices for shared resources to account for network congestion. By solving the local optimization problem to maximize their individual

utility, the system as a whole is able to maximize its realized utility. In our system, because the prices are set by current demand and utilization, there is no benefit to the client to try and "cheat" the pricing scheme, i.e., the prices in the system directly reflect the impacts of congestion on the client's selfish interests.

10.7 Conclusion

In this chapter, we have demonstrated a technique for resource allocation in overlay network based environments that are derived from Lagrangian optimization techniques similar to those used in Internet congestion control. Our approach has some clear advantages over some obvious solutions for routing data within an overlay network. Principally, our decentralized approach is capable of producing a near-optimal assignment, and maintains the more attractive attributes of a decentralized solution, e.g., scalability and reliability. In addition, we use the model of this overlay network to derive a metric suitable for measuring the robustness of this approach.

Throughout this chapter we have assumed that a feasible solution to the centralized allocation problem exists. Future work in this area may explore problems where fluctuations in the production rate of requests results in an infeasible system. Additional research also may include combining the two example environments into a single system.

CHAPTER 11

CONCLUSIONS

In this research, we have demonstrated numerous techniques for resource allocation in a variety of environments that are based on a quantifiable measure of robustness. In each of these environments we have sought to derive a mathematical model of robustness that can be used to evaluate resource allocation decisions as well as guide resource allocation decision making.

The main contributions of this research are (1) a mathematical derivation of robustness suitable for a dynamic environment based on point estimates of system parameters (2) a mathematical definition of robustness applicable to environments where the uncertainty in system parameters can be modeled stochastically, (3) a demonstration of the use of this metric to design resource allocation heuristics in a static environment, (4) a mathematical definition of robustness in a stochastic dynamic environment, (5) a demonstration of the use of this dynamic robustness metric to design resource allocation heuristics suitable for a given heterogeneous computing system, (6) the derivation of a robustness metric for resource allocations in an overlay network along with a near optimal resource allocation technique for this environment.

REFERENCES

[1] S. Ali, T. D. Braun, H. J. Siegel, A. A. Maciejewski, N. Beck, L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, *Characterizing Resource Allocation Heuristics for Heterogeneous Computing Systems*, ser. Advances in Computers. Amsterdam, The Netherlands: Elsevier, 2005, pp. 91–128.

[2] S. Ali, J. K. Kim, Y. Yu, S. B. Gundala, S. Gertphol, H. J. Siegel, A. A. Maciejewski, and V. Prasanna, "Utilization-based techniques for statically mapping heterogeneous applications onto the HiPer-D heterogeneous computing system," *Parallel and Distributed Computing Practices*, vol. 5, no. 4, Dec. 2002.

[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, pp. 630–641, Jul. 2004.

[4] S. Ali, H. J. Siegel, M. Maheswaran, and D. Hensgen, "Representing task and machine heterogeneities for heterogeneous computing systems," *Tamkang Journal of Science and Engineering Special* 50^{th} *Anniversary Issue*, vol. 3, no. 3, pp. 195–207, Nov. 2000.

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

[6] S. Ali, H. J. Siegel, and A. A. Maciejewski, "The robustness of a resource allocation in parallel and distributed computing systems," in *Proceedings of the joint meeting of ISPDC 2004: Third International Symposium on Parallel and Distributed Computing, and HeteroPar '04: Third International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, Jul. 2004, pp. 2–10.

[7] M. Arlitt and T. Jin, "Workload characterization of the 1998 world cup web site," Hewlett Packard Corporation, CA, Tech. Rep. HPL-1999-35R1, Sep. 1999.

[8] ——, "A workload characterization study of the 1998 world cup web site," *IEEE Network*, vol. 14, pp. 30–37, May/Jun. 2000.

[9] M. F. Arlitt and C. L. Williamson, "Internet Web servers: Workload characterization and performance implications," *IEEE/ACM Transactions on Networking*, vol. 5, no. 5, pp. 631–645, Oct. 1997.

[10] (2005) Adaptive and Reflective Middleware Systems (ARMS). Accessed Dec. 10, 2005. [Online]. Available: http://dtsn.darpa.mil/ixo/ixo_FeatureDetail.asp?id=6#

[11] X. Bai, D. C. Marinescu, L. Bölöni, H. J. Siegel, R. A. Daley, and I.-J. Wang, "A macroeconomic model for resource allocation in large-scale distributed systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 2, pp. 182–199, 2008.

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

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

[14] L. Barbulescu, A. E. Howe, L. D. Whitley, and M. Roberts, "Trading places: How to schedule more in a multi-resource oversubscribed scheduling problem," in *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS 2004)*, 2004, pp. 227–234.

[15] L. Barbulescu, L. D. Whitley, and A. E. Howe, "Leap before you look: An effective strategy in an oversubscribed scheduling problem," in *Proceedings of the* 19^th *National Conference on Artifial Intelligence*, 2004, pp. 143–148.

[16] L. A. Barroso, J. Dean, and U. Holzle, "Web search for a planet: The Google cluster architecture," *Micro IEEE*, vol. 23, no. 2, pp. 22–28, Apr. 2003.

[17] J. Bean, J. Birge, J. Mittenthal, and C. Noon, "Matchup scheduling with multiple resources, release dates and disruptions," *Journal of the Operations Research Society of America*, vol. 39, no. 3, pp. 470–483, Jun. 1991.

[18] G. Bernat, A. Colin, and S. M. Peters, "WCET analysis of probabilistic hard real-time systems," in *Proceedings of the* 23^{rd} *IEEE Real-Time Systems Symposium (RTSS '02)*, 2002.

[19] D. P. Bertsekas, *Nonlinear Programming*, 2nd ed. Belmont, MA: Athena Scientific, 2003.

[20] ——, *Dynamic Programming and Optimal Control*, 3rd ed. Belmont, MA: Athena Scientific, 2005.

[21] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, 1st ed. Englewood Cliffs, NJ, USA: Prentice Hall, 1989.

[22] N. Bharadwaj and V. Chandrasekar, "Waveform design for casa x-band radars," in *Proceedings of the* 32^{nd} *Conference on Radar Meteorology of the American Meteorology Society*, Oct. 2005.

[23] K. P. Birman, *Reliable Distributed Systems, Technologies, Web Services, and Applications*, 1st ed. New York, New York: Springer Science+Business Media, 2005.

[24] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed., B. S. M. Jordan, J. Kleinberg, Ed. 233 Spring Street, New York, NY 10013, USA: Springer Science+Business Media, LLC, 2006.

[25] L. Bölöni and D. Marinescu, "Robust scheduling of metaprograms," *Journal of Scheduling*, vol. 5, no. 5, pp. 395–412, Sep. 2002.

[26] T. D. Braun, H. J. Siegel, N. Beck, L. Bölöni, R. F. Freund, D. Hensgen, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, and B. Yao, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, Jun. 2001.

[27] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, "Economic models for resource management and scheduling in grid computing," *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1507–1542, 2002.

[28] R. Castain, W. W. Saylor, and H. J. Siegel, "Application of lagrangian receding horizon techniques to resource management in ad-hoc grid environments," in 13^{th} *Heterogeneous Computing Workshop (HCW 2004), in the Proceedings of the* 18^{th} *International Parallel and Distributed Processing Symposium (IPDPS 2004)*, Apr. 2004.

[29] E. K. P. Chong and B. E. Brewington, "Decentralized rate control for tracking and surveillance networks," *Ad Hoc Networks, special issue on Recent Advances in Wireless Sensor Networks*, vol. 5, no. 6, pp. 910–928, Aug. 2007.

[30] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization*, 2nd ed. New York, NY: John Wiley, 2001.

[31] B. Chun, P. Buonadonna, A. AuYoung, C. Ng, D. Parkes, J. Shneidman, A. Snoeren, and A. Vahdat, "Mirage: A microeconomic resource allocation system for sensornet testbeds," in *Proceedings of The Second IEEE Workshop on Embedded Networked Sensors, 2005. EmNetS-II.*, 30-31 May 2005, pp. 19–28.

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

[33] G. Coullouris, J. Dollimore, and T. Kindberg, *Distributed Systems Concepts and Design*, 4th ed. Harlow, England: Addison Wesley, 2005.

[34] R. L. Daniels and J. E. Carillo, "β-robust scheduling for single-machine systems with uncertain processing times," *IIE Transactions*, vol. 29, no. 11, pp. 977–985, Aug. 1997.

[35] L. David and I. Puaut, "Static determination of probabilistic execution times," in *Proceedings of the* 16^{th} *Euromicro Conference on Real-Time Systems (ECRTS '04)*, Jun. 2004.

[36] A. Dogan and F. Ozguner, "Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems," *Cluster Computing*, vol. 7, no. 2, pp. 177–190, Apr. 2004.

[37] M. Dorigo and L. M. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[38] K. Emo, "The 'enterprise-class' service bus: IT's direct route to SOA," *Business Integration Journal*, Oct. 2005.

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

[40] (2008) f5 network layer load balancer. Accessed Feb. 27, 2008. [Online]. Available: http://www.f5.com

[41] M. Feldman, K. Lai, and L. Zhang, "A price-anticipating resource allocation mechanism for distributed shared clusters," in *EC '05: Proceedings of the 6th ACM conference on Electronic commerce.* New York, NY, USA: ACM Press, 2005, pp. 127–136.

[42] D. Fernandez-Baca, "Allocating modules to processors in a distributed system," *IEEE Transactions on Software Engineering*, vol. 15, no. 11, pp. 1427–1436, Nov. 1989.

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

[44] I. Foster and C. Kesselman, Eds., *The Grid 2: Blueprint for a New Computing Infrastructure.* San Francisco, CA: Morgan Kaufmann, 1999.

[45] R. F. Freund and H. J. Siegel, "Heterogeneous processing," *IEEE Computer*, vol. 26, no. 6, pp. 13–17, Jun. 1993.

[46] A. Ghafoor and J. Yang, "A distributed heterogeneous supercomputing management system," *IEEE Computer*, vol. 26, no. 6, pp. 78–86, Jun. 1993.

[47] N. Gharachorloo, S. Gupta, R. F. Sproull, and I. E. Sutherland, "A characterization of ten rasterization techniques," in *SIGGRAPH '89: Proceedings of the* 16^{th} *Annual Conference on Computer Graphics and Interactive Techniques.* New York, NY, USA: ACM Press, 1989, pp. 355–368.

[48] C. C. Gonzaga, "Path-following methods for linear programming," *SIAM Review*, vol. 34, no. 2, pp. 167–224, Jun. 1992.

[49] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence.* Cambridge, MA: MIT Press, 1992.

[50] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on non-identical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, Apr. 1977.

[51] M. A. Iverson, F. Ozguner, and L. Potter, "Statistical prediction of task execution times through analytical benchmarking for scheduling in a heterogeneous environment," *IEEE Transactions on Computers*, vol. 48, no. 12, pp. 1374–1379, Dec. 1999.

[52] H. A. James, K. A. Hawik, and P. D. Coddington, "Scheduling independent tasks on metacomputing systems," in *Proceedings of the* 12^{th} *International Conference on Parallel and Distributed Computing Systems (PDCS 99)*, Aug. 1999, pp. 47–52.

[53] F. Junyent, V. Chandrasekar, D. McLaughlin, S. Frasier, E. Insanic, R. Ahmed, N. Bharadwaj, E. Knapp, L. Krnan, and R. Tessler, "Salient features of radar nodes of the first generation netrad system," in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium 2005 (IGARSS '05)*, Jul. 2005, pp. 420–423.

[54] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42–51, Jul. 1998.

[55] M. Karaul, Y. A. Korilis, and A. Orda, "A market-based architecture for management of geographically dispersed, replicated web servers," *Decision Support Systems*, vol. 28, no. 1-2, pp. 191–204, Mar. 2000.

[56] A. Khokhar, V. K. Prasanna, M. E. Shaaban, and C. Wang, "Heterogeneous computing: Challenges and opportunities," *IEEE Computer*, vol. 26, no. 6, pp. 18–27, Jun. 1993.

[57] J.-K. Kim, D. A. Hensgen, T. Kidd, H. J. Siegel, D. S. John, C. Irvine, T. Levin, N. W. Porter, V. K. Prasanna, and R. F. Freund, "A flexible multi-dimensional qos performance measure framework for distributed heterogeneous systems," *Cluster Computing, Special Issue on Cluster Computing in Science and Engineering*, vol. 6, no. 3, pp. 281–296, Jul. 2006.

[58] J. K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S. S. Yellampalli, "Dynamic mapping in a heterogeneous environment with tasks having priorities and multiple deadlines," in 12^{th} *Heterogeneous Computing Workshop (HCW 2003), in the Proceedings of the* 17^{th} *International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Apr. 2003.

[59] A. Kumar and R. Shorey, "Performance analysis and scheduling of stochastic fork-join jobs in a multicomputer system," *Transactions on Parallel and Distributed Systems*, vol. 4, no. 10, Oct. 1993.

[60] K. Lai, L. Rasmusson, E. Adar, L. Zhang, and B. A. Huberman, "Tycoon: an implementation of a distributed, market-based resource allocation system," *Multiagent and Grid Systems*, vol. 1, no. 3, pp. 169–182, 2005.

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

[62] V. J. Leon, S. D. Wu, and R. H. Storer, "Robustness measures and robust scheduling for job shops," *IIE Transactions*, vol. 26, no. 5, pp. 32–43, Sep. 1994.

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

[64] Y. A. Li, J. K. Antonio, H. J. Siegel, M. Tan, and D. W. Watson, "Determining the execution time distribution for a data parallel program in a heterogeneous computing environment," *Journal of Parallel and Distributed Computing*, vol. 44, no. 1, pp. 35–52, Jul. 1997.

[65] P. Luh, X. Zhao, Y. Wang, and L. Thakur, "Lagrangian relaxation neural networks for job shop scheduling," *IEEE Transactions on Robotics and Automation*, vol. 16, no. 1, pp. 78–88, Feb. 2000.

[66] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–121, Nov. 1999.

[67] M. Maheswaran, T. D. Braun, and H. J. Siegel, *Heterogeneous distributed computing*, ser. Encyclopedia of Electrical and Electronics Engineering. New York, NY: John Wiley & Sons, 1999, vol. 8, pp. 679–690.

[68] G. Mainland, D. C. Parkes, and M. Welsh, "Decentralized, adaptive resource allocation for sensor networks," in *NSDI'05: Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation.* Berkeley, CA, USA: USENIX Association, 2005, pp. 315–328.

[69] P. Manghwani, J. Loyall, P. Sharma, M. Gillen, and J. Ye, "End-to-end quality of service management for distributed real-time embedded applications," in *Proceedings of the* 19^{th} *IEEE International Parallel and Distributed Processing Symposium (IPDPS '05)*, Apr. 2005.

[70] A. Mehta, J. Smith, H. J. Siegel, A. A. Maciejewski, A. Jayaseelan, and B. Ye, "Dynamic resource allocation heuristics for maximizing robustness with an overall makespan constraint in an uncertain environment," in *Proceedings of The 2006 International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. 1, Jun. 2006, pp. 24–30.

[71] ——, "Dynamic resource management heuristics for minimizing makespan while maintaining an acceptable level of robustness," in *Proceedings of The* 12^{th} *International Conference on Parallel and Distributed Systems (ICPADS 2006)*, Jul. 2006, pp. 107–114.

[72] A. M. Mehta, J. Smith, H. J. Siegel, A. A. Maciejewski, A. Jayaseelan, and B. Ye, "Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness," *Journal of Supercomputing, Special Issue on Grid Technology*, vol. 42, no. 1, pp. 33–58, Oct. 2007.

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

[74] K. G. Murty and S. N. Kabadi, "Some NP-complete problems in quadratic and nonlinear programming," *Mathematical Programming*, vol. 39, no. 2, pp. 117–129, Nov. 1987.

[75] V. K. Naik, S. Sivasubramanian, D. Bantz, and S. Krishnan, "Harmony: A desktop grid for delivering enterprise computations," in *Proceedings of the Fourth International Workshop on Grid Computing (GRID '03)*, Nov. 2003.

[76] (2007) Opnet technologies, inc. Accessed Feb. 20, 2007. [Online]. Available: http://www.opnet.com/

[77] A. J. Page and T. J. Naughton, "Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing," in *Proceedings of the* 19^{th} *International Parallel and Distributed Processing Symposium*, Apr. 2005.

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

[79] G. Pierre and M. van Steen, "Globule: a collaborative content delivery network," *IEEE Communications Magazine*, vol. 44, no. 8, pp. 127–133, Aug. 2006.

[80] N. Policella, "Scheduling with uncertainty, a proactive approach using partial order scheduling," Ph.D. dissertation, Universit a degli Studi di Roma, La Sapienza, 2005.

[81] C. R. Reeves and T. Yamada, "Genetic algorithms, path relinking and the flowshop sequencing problem," *Evolutionary Computation Journal*, vol. 6, no. 1, pp. 230–234, 1998.

[82] K. Ross and N. Bambos, "Local search scheduling algorithms for maximal throughput in packet switches," in *Proceedings of the* 23^{rd} *Conference of the IEEE Communications Society (INFOCOM 2004)*, 2004.

[83] M. T. Schmidt, B. Hutchison, P. Lambros, and R. Phippen, "The enterprise service bus: Making service-oriented architecture real," *IBM Systems Journal*, vol. 44, no. 4, pp. 781–797, 2005.

[84] V. Shestak, E. K. P. Chong, A. A. Maciejewski, H. J. Siegel, L. Benmohamed, I. J. Wang, and R. Daley, "Resource allocation for periodic applications in a shipboard computing environment," in 14^{th} *Heterogeneous Computing Workshop (HCW 2005)*

in Proceedings of the 19^{th} *International Parallel and Distributed Processing Symposium (IPDPS 2005)*, Apr. 2005, pp. 122–127.

[85] V. Shestak, J. Smith, H. J. Siegel, and A. A. Maciejewski, "Iterative algorithms for stochastically robust static resource allocation in periodic sensor driven clusters," in *Proceedings of the* 18^{th} *IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2006)*, Nov. 2006, pp. 166–174.

[86] ——, "A stochastic approach to measuring the robustness of resource allocations in distributed systems," in *Proceedings of the 2006 International Conference on Parallel Processing (ICPP 2006)*, Aug. 2006, pp. 6–12.

[87] V. Shestak, J. Smith, R. Umland, J. Hale, P. Moranville, A. A. Maciejewski, and H. J. Siegel, "Greedy approaches to static stochastic robust resource allocation for periodic sensor driven systems," in *Proceedings of The 2006 International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. 1, Jun. 2006, pp. 3–9.

[88] V. Shestak, E. K. P. Chong, H. J. Siegel, A. A. Maciejewski, L. Benmohamed, I. Wang, and R. Daley, "A hybrid branch-and-bound and evolutionary approach for allocating strings of applications to heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 68, no. 4, pp. 410–426, Apr. 2008.

[89] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Stochastic robustness metric and its use for static resource allocations," *Journal of Parallel and Distributed Computing*, accepted to appear, 2008.

[90] S. Shivle, H. J. Siegel, A. A. Maciejewski, P. Sugavanam, T. Banka, R. Castain, K. Chindam, S. Dussinger, P. Pichumani, P. Satyasekaran, W. Saylor, D. Sendek, J. Sousa, J. Sridharan, and J. Velazco, "Static allocation of resources to communicating subtasks in a heterogeneous ad-hoc grid environment," *Journal of Parallel and Distributed Computing, Special Issue on Algorithms for Wireless and Ad-hoc Networks*, vol. 66, no. 4, pp. 600–611, Apr. 2006.

[91] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," in *Proceedings of the* 5^{th} *IEEE Heterogeneous Computing Workshop (HCW '96)*, 1996, pp. 86–97.

[92] S. Sivisubramanian, B. van Halderen, and G. Pierre, "Globule: A user-centric content delivery network," in *Proceedings of the* 4^{th} *International Systems Administration and Network Engineering Conference (SANE 2004)*, 2004.

[93] J. Smith, L. D. Briceño, A. A. Maciejewski, and H. J. Siegel, "Measuring the robustness of resource allocations in a stochastic dynamic environment," in *Proceedings of the* 21^{st} *International Parallel and Distributed Processing Symposium (IPDPS 2007)*, Mar. 2007.

[94] J. Smith, E. K. P. Chong, A. A. Maciejewski, and H. J. Siegel, "Decentralized market-based resource allocation in a heterogeneous computing system," in *Proceedings of the* 22^2*nd International Parallel and Distributed Processing Symposium*, Apr. 2008.

[95] J. Smith, V. Shestak, H. J. Siegel, S. Price, L. Teklits, and P. Sugavanum, "Resource allocation in a cluster based imaging system," in *Proceedings of the 2007 International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. 1, Jun. 2007, pp. 3–9.

[96] J. Smith, H. J. Siegel, and A. A. Maciejewski, "Iterative techniques for maximizing stochastic robustness of a static resource allocation in periodic sensor driven clusters," in *Proceedings of the 2008 International Conference on Parallel and Distributed Processing Techniques and Applications*, accepted, to appear.

[97] ——, *Robust Resource Allocation in Heterogeneous Parallel and Distributed Computing Systems*, ser. Wiley Encyclopedia of Computing. New York, NY: John Wiley & Sons, accepted to appear.

[98] (2006) Standard performance evaluation corporation. Accessed Feb. 2006. [Online]. Available: http://www.spec.org/

[99] R. Srikant, *The Mathematics of Internet Congestion Control.* Boston, MA: Birkhauser, 2003.

[100] 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*, vol. 67, no. 4, pp. 400–416, Apr. 2007.

[101] X. Tang and S. T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," in *Proceedings of the International Conference on Parallel Processing 2000 (ICPP'00)*, Aug. 2000, p. 373.

[102] R. Vaessens, E. Aarts, and J. Lenstra, "Job-shop scheduling by local search," Eindhoven University of Technology, Eindhoven, The Netherlands, Tech. Rep. http://citeseer.ist.psu.edu/vaessens94job.html, 1994 (accessed May 2006).

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

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

[105] D. Whitley, "The genitor algorithm and selective pressure: Why rank-based allocation of reproductive trials is best," in *Proceedings of the* 3^{rd} *International Conference on Genetic Algorithms*, Jun. 1989, pp. 116–121.

[106] L. A. Wolsey, *Integer Programming.* New York, NY: John Wiley & Sons, 1998.

[107] M. Wu, W. Shu, and H. Zhang, "Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems," in *Proceedings of the* 9^{th} *IEEE Heterogeneous Computing Workshop*, Mar. 2000, pp. 375–385.

[108] D. Xu, K. Nahrstedt, and D. Wichadakul, "Qos and contention-aware multi-resource reservation," *Cluster Computing*, vol. 4, no. 2, pp. 95–107, Apr. 2001.

[109] J. Yang, I. Ahmad, and A. Ghafoor, "Estimation of execution times on heterogeneous supercomputer architectures," in *Proceedings of the International Conference on Parallel Processing*, Aug. 1993, pp. 219–226.

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