DISSERTATION

Robust Resource-Allocation Methods for QOS-Constrained Parallel and Distributed Computing Systems

Submitted by

Vladimir Shestak

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2008

COLORADO STATE UNIVERSITY

November 11, 2008

WE HEREBY RECOMMEND THAT THE **DISSERTATION** PREPARED UNDER OUR SUPERVISION BY **VLADIMIR SHESTAK CANDIDATE** ENTITLED **ROBUST RESOURCE-ALLOCATION METHODS FOR QOS-CONSTRAINED PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS** BE ACCEPTED AS FULFILLING IN PART REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY.

Committee on Graduate Work

Edwin Chong

Charles Anderson
(Department of Computer Science)

Howard Jay Siegel
Co-Adviser

Anthony Maciejewski
Co-Adviser

Department Head/Director

ABSTRACT OF DISSERTATION

ROBUST RESOURCE-ALLOCATION METHODS FOR QOS-CONSTRAINED PARALLEL AND DISTRIBUTED COMPUTING SYSTEMS

This research investigates the problem of robust resource allocation for distributed computing systems operating under imposed Quality of Service (QoS) constraints. Often, such systems are expected to function in a physical environment replete with uncertainty, which causes the amount of processing required over time to fluctuate substantially. In the first two studies, we show how an effective resource allocation can be achieved in the heterogeneous shipboard distributed computing system and IBM cluster based imaging system. The general form for a stochastic robustness metric is then presented based on a mathematical model where the relationship between uncertainty in system parameters and its impact on system performance are described stochastically. The utility of the established metric is exploited in the design of optimization techniques based on greedy and iterative approaches that address the problem of resource allocation in a large class of distributed systems operating on periodically updated data sets. One of the major reasons for possible QoS violations in distributed systems is a loss of resources, frequently caused by abnormal operating conditions. One aspect that makes a resource allocation problem extremely challenging in such systems is a random nature of resource failures and recoveries. The last study presented in this work describes a solution method that was developed for this case based on the concepts of the Derman-Lieberman-Ross theorem. The experimental results indicate a significant potential of this approach to generate robust resource allocations in unstable distributed systems.

Vladimir Shestak Candidate
Department of Electrical and Computer Engineering
Colorado State University
Fort Collins, CO 80523
Fall 2008

Contents

List of Tables

List of Figures

Acknowlegements

I would like to express my gratitude to my supervisors, Professors H.J. Siegel, Anthony Maciejewski, and Edwin Chong for patiently guiding me through my graduate experience. I have learned invaluable research and organizational skills from them. The high standards they required of me furthered my knowledge and career. I would also like to thank Professor Charles Anderson for taking time out from his busy schedule to provide the helpful comments and guidance.

I must thank the InfoPrint Solutions Comppany (former IBM Printing Systems Division) for the opportunity to apply knowledge I obtained in academia in the state-of-the-art industrial environment. Industry experts Larry Teklits, Larry Ernst, Suzy Price, and Joan Mitchel brought so much good into my life. It is a great honor for me to continiue working with them after my graduation.

I would also like to thank my friends in the research group, particularly James Smith, for enriching exchanges of knowledge and skills during my graduate program.

A very special thanks goes out to my wife, Irina Shestak, for her patience and understanding, and to my mother and father, Svetlana and Vladimir Shestak, for their support and encouragement in my graduate career.

This research was supported by InfoPrint Solutions Company, the National Science Foundation under grant No. CNS-0615170, the DARPA Information Exploitation Office under contract No. NBCHC030137, the IBM Ph.D. Fellowship Program for 2006–2007 academic year, the Colorado State University Center for Robustness in Computer Systems (funded by the Colorado Commission on Higher Education Technology Advancement Group through the Colorado Institute of Technology), and the Colorado State University George T. Abell Endowment.

Chapter 1

Introduction

Often, parallel and distributed computing systems must operate in an environment replete with uncertainty while providing a required level of quality of service (QoS). The robust design in such systems is becoming an increasingly important issue as it is demonstrated in the following examples [7]. 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 organizations. This thesis addresses, for the allocation of computing and communication resources in a parallel and distributed system, the problems of developing a generalized stochastic robustness metric, deriving robust resource allocations, and maximizing system performance

under random resource failures and recoveries.

The design scheme developed in this research for resource allocations in QoS-constrained distributed systems operating under uncertainty includes the following major steps:

1. Establish a metric that quantifies system performance with respect to the imposed QoS constraints.

2. Develop a mathematical model that provides a functional dependence between the performance metric, input parameters, and uncertainties in the system.

3. Integrate this model into an adapted or developed optimization technique. Due to the typical NP-complete nature of the resource allocation problem in heterogeneous systems, an optimization technique is usually a heuristic or a mathematical method that results in a sub-optimal solution.

4. Evaluate the quality of a sub-optimal solution and compare it against sub-optimal solutions of other optimization techniques. However, this comparison analysis provides only a relative performance evaluation. An absolute evaluation can be obtained by comparing a sub-optimal solution against a performance bound.

The work described in Chapter 2 was based on the problem statement for the research supported by the DARPA Information Exploitation Office, under the project called "Adaptive and Reflective Middleware Systems (ARMS)." It involved the design and analysis of a heuristic that allocates computation and communication resources to the strings of applications in a complex shipboard computing system. The considered system consists of a set of dedicated machines interconnected by high-speed

communication links. A set of sensors (radars, sonars, etc.) sends streams of data sets to a set of communicating, continuously running applications organized in strings that process these data sets and send their outputs to other applications or actuators. When running, the ARMS system is required to satisfy a set of throughput and latency constraints. Any allocation of the resources must enforce these quality of service (QoS) constraints, i.e., it must ensure that the computation and communication times are within certain limits. When the ship leaves a dock, its equipment is assumed to be functional and operating under the nominal values of sensor loads (i.e., outputs from sensors). However, the system is expected to operate in a dynamic environment, where the sensor loads are expected to change unpredictably. Increases in sensor loads cause increases in the computation and communication times, which in turn may cause throughput and latency violations. Therefore, an initial resource allocation designed for the ARMS system must be able to tolerate as much sensor load increase as possible before a QoS violation occurs. A two-stage approach was designed to solve this problem based on a combination of the evolutionary and greedy heuristics in the first stage and a Branch-and-Bound algorithm in the second stage. Additional contributions include developing the application and hardware models of the considered part of the shipboard environment, quantifying the performance goals for different scenarios, evaluating the relative performance of the heuristics developed, and deriving mathematical bounds on performance based on a Linear Programming relaxation method.

Chapter 3 in this thesis presents the research done for the InfoPrint Solution Company (former IBM Printing Systems Division). This is another example of a QoS-constrained distributed system. In this case, image processing is performed under uncertainty. Similar to the ARMS example, the system must satisfy a certain level of

QoS, i.e., the output generated in the system must be delivered to the raster-based displays at regular intervals, effectively establishing a hard deadline for the completion of each output image. The challenge in the design of the resource allocation comes not only from the uncertain times required to rasterize images, but also from the communication and memory sharing issues, as these issues significantly complicate the completion time estimation process. Furthermore, the desired resource allocation must guarantee a sustainable system performance with a minimum set of hardware resources. The primary contributions of this chapter 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 first part of the research presented in Chapter 4 proposes a generalized framework where the performance metric, input parameters, and uncertainties in a system are treated as random variables. Consequently, all operations with these variables are carried out in the stochastic domain. As it is shown in our experiments, the proposed stochastic framework improves the accuracy of the performance metric but it comes at an additional computational expense. The utility of the new framework was evaluated in the second part of that research that covers the last two items of the list above. Four greedy and three global search heuristics were adapted to address the problem of a resource allocation in the simulated environment based on two algorithms used in the Collaborative Adaptive Sensing of the Atmosphere (CASA) system. A performance lower bound was derived analytically by relaxing the Integer Linear Programming form.

One of the major reasons for possible QoS violations in distributed systems is a loss of resources. Quite often, modern systems experience temporal resource failures mostly caused by abnormal operating conditions. One aspect that makes a resource allocation problem extremely challenging in such systems is a random process of resource failures and recoveries. To maximize the performance of a system, a resource allocation needs to be generated by considering a system's current and possible future states. Although, the fault tolerant aspect of distributed computing systems has been extensively explored in the last few years, the available literature in such uncertain environments primarily considers reactive techniques, where a node failure is addressed only after its occurrence. Checkpoint-resume or terminate-restart mechanisms are often used to recover unprocessed tasks at the failed nodes. Node failure also can be addressed by keeping multiple copies of the workload on different nodes. These approaches are coupled in practice with redundancy schemes that duplicate system hardware resources entirely or partially. Depending on the implementation, duplicated resources are either always active or become active dynamically. Additionally, most of the existing literature that offers an analytical formulation of distributed-computing systems assumes a homogeneity among compute nodes and known system parameters. Our solution method presented in Chapter 5, developed based on the concepts of the Derman-Lieberman-Ross theorem, utilizes the available stochastic information. Mapping decisions are made sequentially: (1) at each time when a decision-maker observes a type of the arriving processor, i.e., the realization of the random variable, (2) it must select a task, i.e., an action, such that the expected cumulative reward is maximized. The experimental results, compared against some commonly used policies, indicate a significant potential of this approach to generate robust resource allocations in unstable distributed systems.

Chapter 2

A Two-Stage Approach to Resource Allocation for Periodic Strings of Applications

2.1 Overview

Providing efficient workload management is an important issue for a large-scale heterogeneous distributed computing environment where a set of periodic applications is executed. The considered shipboard distributed system is expected to operate in an environment where the input workload is likely to change unpredictably, possibly invalidating a resource allocation that was based on the initial workload estimate. The tasks consist of multiple strings, each made up of an ordered sequence of applications. There is a quality of service (QoS) minimum throughput constraint that must be satisfied for each application in a string, and a maximum utilization constraint that must be satisfied on each of the hardware resources in the system. The challenge, therefore,

is to efficiently and robustly manage both computation and communication resources in this unpredictable environment to achieve high performance while satisfying the imposed constraints. This work addresses the problem of finding a robust initial allocation of resources to strings of applications that is able to absorb some level of unknown input workload increase without rescheduling. The proposed two-stage method of finding a near-optimal allocation of resources incorporates two specially designed mapping techniques: (1) the Permutation Space Genitor-Based heuristic, and (2) the follow-up Branch-and-Bound heuristic based on an Integer Linear Programming (ILP) problem formulation. The performance of the proposed resource allocation method is evaluated under different simulation scenarios and compared to an iteratively computed upper bound.

2.2 Introduction and Problem Statement

The research described in this chapter investigates the problem of robust static resource allocation for shipboard computing resources in the Adaptive and Reflective Middleware Systems (ARMS) program supported by the DARPA Information Exploitation Office [9]. In this chapter, a resource allocation problem is addressed for a part (due to undisclosed content) of the proposed shipboard environment, depicted schematically in Fig. 2.1, where a limited subset of the ARMS application model is considered. As Fig. 2.1 shows, the target system consists of a number of sensors generating raw data forwarded to the heterogeneous distributed computing system for processing. The computing system itself is composed of a set of machines of various types, a communication network, and continuously running applications processing data coming from the sensors. Data processing in the system must be done by a

sequence of applications in a pipeline fashion; this requirement imposes a quality of service constraint (QoS) on each application's processing time and each internal data transfer's time between applications.

The system is configured with an initial *mapping* (i.e., an allocation of computing and networking resources to applications) that is used when the system is first put into operation. The system is expected to function in an uncertain physical environment where the workload, i.e., the load presented by a set of sensors, is likely to change unpredictably over time, possibly causing a QoS violation. When this occurs, resources in the system need to be reallocated reactively, which results in a temporal performance degradation and, thus, is highly undesirable. Therefore, the general focus of this study is on developing a resource allocation technique to determine a *robust* mapping capable of absorbing the maximum increase in workload without a run-time reallocation of resources. Reallocation techniques are outside the scope of this chapter, but a variety of them can be found in the literature (e.g., [45, 78]).

Specifically, two resource allocation scenarios are addressed in this chapter that differ in their performance goals. A partial resource allocation scenario occurs in an *oversubscribed* system where one or more sequences of applications considered for mapping cannot be allocated due to the limited system resources or predicted QoS constraint violations. Given such a scenario, the primary performance goal for a mapper is to find a static (i.e., one found during an off-line planning phase) initial mapping maximizing a "total worth" of the workload processed. In contrast, a complete resource allocation scenario is relevant for a system that has enough resources to accommodate all applications considered without violating any of the imposed QoS constraints. In this case, a "system slackness" metric reflecting the system's capacity to absorb workload surges is a major optimization criterion for a static mapping.

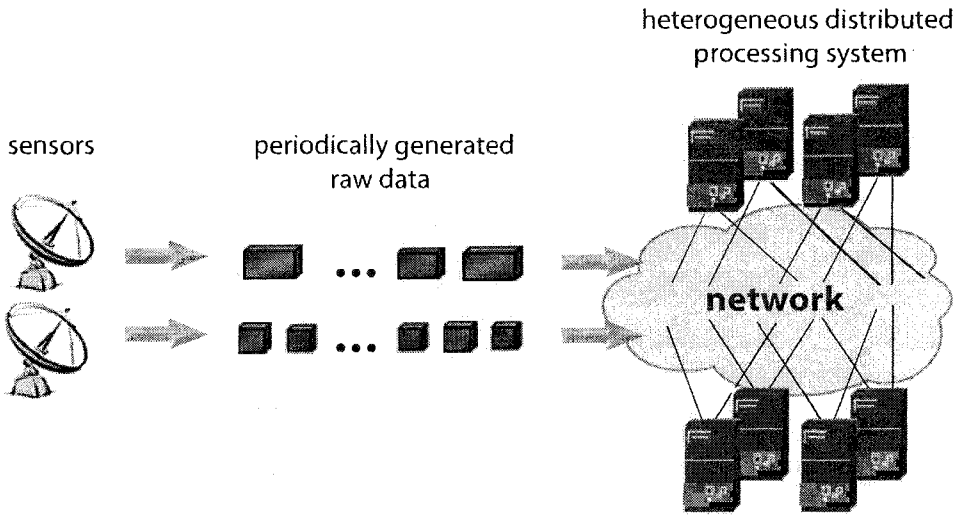


Figure 2.1: The considered part of the ARMS shipboard environment. Each sensor generates raw data periodically and forwards it to the distributed heterogeneous processing system.

Given that the problem of resource allocation in distributed systems is NP-complete (e.g., [21, 49]), the development of heuristic techniques to find near-optimal solutions became an active area of research (e.g., [5, 16, 17, 37, 77, 99]). In general, there are two major classes of resource allocation approaches widely used in practice: greedy heuristics and global optimization algorithms. Unlike rather time-consuming global optimization algorithms, greedy heuristics are relatively fast in generating a single solution; this feature often makes them an appropriate choice to use in dynamic (i.e., on-line mapping) systems. However, the quality of solutions based on greedy heuristics is usually lower than that produced by global optimization algorithms that progressively iterate through multiple solutions.

One type of global optimization algorithms is a class of evolutionary techniques. In principle, such techniques rely on an "intelligent" randomized search where a solution

is picked in the solution space, and its fitness is then evaluated. The efficiency of this method often suffers when applied to constrained optimization problems because of the time wasted generating infeasible solutions. To resolve this issue, the chapter proposes an approach in the first stage of finding a resource allocation where the search space of the evolutionary algorithm differs from the actual solution space, and a specifically constructed greedy heuristic is used to link these two spaces.

Another drawback of evolutionary algorithms is a lack of structural organization of the underlying search process. For problems of realistic size, it is highly unlikely that randomized search will find a global optimal solution in a reasonable amount of time. Furthermore, even if an algorithm converges to the global optimal solution it has no means of proving it. To overcome this problem, a two-stage approach in this study proposes that the final solution of the evolutionary algorithm is passed to a Branch-and-Bound technique based on an Integer Linear Programming formulation of the resource allocation problem. The key point is that the well structured tree search in the Branch-and-Bound algorithms becomes significantly more efficient when a high-quality solution is received that can be used for pruning the search tree. Furthermore, a backtracking mechanism in our second-stage Branch-and-Bound algorithm allows the upper bound on performance metrics to be tightened as the algorithm progresses.

In addition to the proposed two-stage approach to resource allocation utilizing a combination of the evolutionary and greedy heuristics in the first stage and a Branch-and-Bound algorithm in the second stage, the contributions of this work include developing the application and hardware models of the considered part of the shipboard environment, quantifying the performance goals for different scenarios, evaluating the relative performance of the heuristics developed, and deriving mathematical bounds on performance based on a Linear Programming relaxation method.

The remainder of this chapter is organized in the following manner. Section 2.3 develops models for the workload and hardware platform. Section 2.4 presents a quantitative basis for the performance measure for a given resource allocation. In Section 2.5, the Genitor-based evolutionary algorithm is described along with a special-purpose greedy mapping routine developed to generate solutions for both allocation scenarios in the first stage. A mathematical model for finding performance upper bounds in different scenarios based on a Linear Programming relaxation is provided in Section 2.6. Section 2.7 presents a set of Branch-and-Bound algorithms developed to improve on the first stage resource allocations and tighten the upper bounds. The simulation setup, results, and performance evaluation of the heuristics are discussed in Section 2.8. A sampling of some related work is presented in Section 2.9. Section 2.10 concludes the chapter. A glossary of notation and acronyms used in the chapter are tabulated in Table 2.1 and Table 2.2, respectively.

2.3 System Model

The considered distributed system is composed of a number of heterogeneous computational resources distributed across a shipboard environment and connected by a communication network.

The functionality of the communication network is modeled by all possible independent virtual point-to-point communication routes, each characterized by a maximum available bandwidth. Existing networking technologies can enforce this communication model through resource reservations at system initialization time. Each machine in the system is capable of multitasking. Similarly, a given communication

Table 2.1: Glossary of Notation

S^k	k^{th} string specified by a sequence of n_k applications $\{a_1^k a_2^k ... a_{n_k}^k\}$
$W[k]$	worth factor of k^{th} string
$P[k]$	period of time between sequential raw data sets processed by k^{th} string
$m[i,k]$	machine to which application a_i^k is assigned
$t_{comp}^k[i]$	estimated computation time for application a_i^k on machine $m[i,k]$
$t_{tran}^k[i]$	estimated time to transfer output $O^k[i]$ from a_i^k to a_{i+1}^k
$\overline{t}^k[i,j]$	nominal data set processing time of a_i^k executing on machine j
$\overline{u}^k[i,j]$	average CPU utilization of machine j when a_i^k processes a nominal data set
$U^{machine}[j]$	utilization of machine j
$b[j_1,j_2]$	time to transmit one bit of data from machine j_1 to machine j_2
$U^{route}[j_1,j_2]$	utilization of the comm. route from machine j_1 to machine j_2
M	number of heterogeneous machines in the system
Λ	system slackness, i.e., the minimum utilization capacity remaining across all computation and communication resources
$\overline{t}_{av}^k[i]$	average nominal execution time of a_i^k across M machines
$\overline{u}_{av}^k[i]$	average nominal CPU utilization of a_i^k across M machines
Q	total number of strings considered for mapping

route is shared among multiple active data transmissions traversing that communication route.

In the given shipboard environment, a string is defined as a continuously executing sequence of applications connected in precedence order by specified data transfers. Data is received by an application from the preceding application, or from a sensor that generates data sets with a fixed period. The output produced by the string serves as an input to other applications or to actuators.

Let S^k be the k^{th} string, specified by a sequence of n_k applications: $S^k = a_1^k a_2^k ... a_{n_k}^k$. To model the importance of each string in the system, for each k, the k^{th} string is preassigned one of three possible worth factors, $W[k] \in \{1,2,3\}$. Worth factors play a significant role in the partial allocation scenario where a subset of strings with the maximum total worth needs to be selected to be deployed in the system.

Table 2.2: Acronyms

ARMS	Adaptive and Reflective Middleware Systems
IMR	Incremental Mapping Routine
PSG	Permutation Space Genitor-based heuristic
ILP	Integer Linear Programming
LP	Linear Programming (can be achieved by relaxing an integer restriction in the corresponding ILP form)
UB	Upper Bound
B&B	Branch-and-Bound algorithm

Let $P[k]$ be the period associated with string S^k, where each a_i^k must execute once each period. The minimum throughput QoS constraint requires that the computation time of any application or the time of any inter-application data transfer in S^k be no larger than $P[k]$. Such an enforcement allows each string to process data in a *pipeline* fashion resulting in high processing efficiency for the entire system. Assuming that a resource allocation for string S^k is made, let $m[i,k]$ denote the machine to which application a_i^k is assigned. Let $t_{comp}^k[i]$ be the estimated computation time for application a_i^k for processing a *nominal* data set (executing on $m[i,k]$). A nominal data set is a data set of mean complexity which is determined based on past executions. Let $t_{tran}^k[i]$ be the estimated transfer time required to send the output of *nominal* size $O^k[i]$ from application a_i^k (on $m[i,k]$) to application a_{i+1}^k (on $m[i+1,k]$) within string S^k. A typical allocation of string S^k in the system is illustrated in Fig. 2.2 below.

Mathematically, for a given resource allocation for a string, the aforementioned minimum throughput QoS constraint is satisfied if:

$$\begin{cases} t_{comp}^k[i] \leq P[k], & 1 \leq i \leq n_k \\ t_{tran}^k[i] \leq P[k], & 1 \leq i \leq n_k - 1 \end{cases} \tag{2.1}$$

If all conditions in (2.1) hold for a given allocation, the allocation is said to be feasible

with respect to the minimum throughput QoS constraint. Because both machines and communication routes are assumed to be shared, $t^k_{comp}[i]$ and $t^k_{tran}[i]$ will depend on the level of sharing – i.e., the number of applications assigned to a computational resource and currently active, or the number of current data transfers assigned to a communication route. Furthermore, these values will depend on how an application or a data transfer is prioritized by a machine's or network's local scheduler with respect to all other applications or data transfers that share this computation or communication resource. In addition to the minimum throughput QoS constraint imposed on strings, the overall utilization of each computation or communication resource must not exceed its full capacity when the system is loaded. Evaluating that the utilization and minimum throughput QoS constraints are satisfied is integrated into the mapping techniques presented in the following sections.

Two parameters are used in the given shipboard environment to specify the workload imposed by each application on a particular machine: the nominal execution time and the nominal CPU utilization. The nominal execution time $\overline{t}^k[i,j]$ is the time required by application a^k_i in string S^k to process a nominal data set on machine j running in non-multitasking mode. Due to the multitasking environment, $t^k_{comp}[i] \geq \overline{t}^k[i, m[i,k]]$. The nominal CPU utilization $\overline{u}^k[i,j]$ is the average

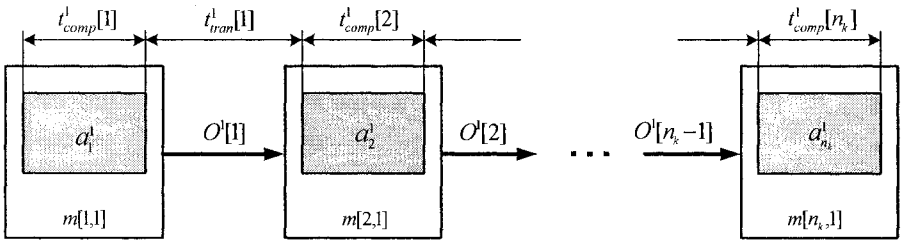


Figure 2.2: The string model for string 1. Shaded rectangles denote applications in the string while white rectangles represent the machines where these applications execute. The arrows represent output data transfers within the string.

CPU utilization of machine j when a_i^k executes its nominal data set. The product $\overline{t}^k[i,j] \times \overline{u}^k[i,j]$ can be interpreted as the fixed amount of CPU work required for application a_i^k to process a nominal data set on machine j. This fixed amount of CPU work can be performed in many different ways. For example, if only half of $\overline{u}^k[i,j]$ is allocated, then the execution time required to accomplish the same fixed amount of CPU work is twice $\overline{t}^k[i,j]$.

Let the conditional **1** function be defined by: $\mathbf{1}(condition) \equiv \begin{cases} 1, & \text{if true} \\ 0, & \text{otherwise} \end{cases}$.

If A strings are allocated in the system then the overall machine utilization $U^{machine}[j]$ is computed as:

$$U^{machine}[j] = \sum_{k=1}^{A} \sum_{i=1}^{n_k} \left(\frac{\overline{t}^k[i,j]}{P[k]} \times \overline{u}^k[i,j] \times \mathbf{1}(m[i,k] = j) \right) \tag{2.2}$$

The term $\overline{t}^k[i,j] \times \overline{u}^k[i,j]/P[k]$ represents the average CPU utilization allocated for application a_i^k over $P[k]$. It is important to note that this is the *minimum* required average CPU utilization that allows a_i^k to complete processing without a throughput QoS constraint violation. Recall from (2.1) that a data set processing time of each application in string S^k must be less or equal to $P[k]$.

The sum of such minimum CPU utilizations across all the applications executing on j determines the overall machine utilization.

If $b[j_1, j_2]$ denotes the time required to transmit one bit of data over the communication route from machine j_1 to machine j_2 then the overall communication route

utilization $U^{route}[j_1, j_2]$ is:

$$U^{route}[j_1, j_2] = b[j_1, j_2] \times \sum_{k=1}^{A} \sum_{i=1}^{n_k - 1} \left(\frac{O^k[i]}{P[k]} \times \mathbf{1}(m[i,k] = j_1 \,\&\, m[i+1,k] = j_2) \right) \quad (2.3)$$

The term $O^k[i]/P[k]$ can be interpreted as the *minimum* average bandwidth allocated to application a_i^k for output transfer over $P[k]$ that allows it to be completed without a minimum throughput QoS constraint violation.

For a given allocation, if the utilization values computed with (2.2) and (2.3) are not greater than one for each machine and communication route then the system is considered to be operating in a feasible (not overloaded) mode.

2.4 Performance Goal

In the context of the intended system, the performance metric for evaluating an application-to-machine mapping generated by the heuristics has two components. The primary component is total worth, defined as the sum of the worth factors associated with strings in the mapping. The secondary component is system slackness. Assuming M machines in the system, let system slackness Λ be a measure of the minimum utilization capacity remaining across all computation and communication resources:

$$\Lambda = \min\left(\{1 - U^{machine}[j] : 1 \le j \le M\} \cup \{1 - U^{route}[j_1, j_2] : 1 \le j_1, j_2 \le M\}\right) \quad (2.4)$$

According to [6], a resource allocation is defined to be robust with respect to specified system performance features against uncertainties in specified system parameters

if degradation in these features is limited. In this work, the system is considered robust if it is able to absorb limited unpredictable changes in input workload which increase resource utilizations without revising a given resource allocation. System slackness is used as a quantitative measure of robustness. The goal of the mapping heuristics developed in this research is to achieve the highest level for the primary component, and then maximizing system slackness Λ at that level.

With the given "worth" scheme, a high-worth string has the same value as three low-worth strings. A different, alternate scheme is possible, where a high-worth string has a value of more than the total value of any number of strings of medium or low worth. In such a scheme, high-worth strings can be put in a special class. The content of this class is allocated first in the system. Such a scheme, described in [55], is outside the current requirements of this work.

2.5 Basic Evolutionary Mapping Algorithm

2.5.1 Overview

This section presents an evolutionary mapping algorithm used as a basis for the problem of finding an initial static mapping in the complete and partial allocation scenarios. To explain the main idea used in the algorithm's development, the following notation needs to be introduced. Let the permutation space be all possible orderings of the strings considered for mapping, and let the solution space be all possible application-to-machine assignments. Recall that an allocation is considered feasible for deployment if none of the computation or communication hardware resources is loaded beyond its maximum utilization capacity.

It was observed experimentally that the straightforward implementations of evolutionary algorithms, e.g., a genetic algorithm [99], operating in the solution space, failed to find any feasible allocation even for a relatively small set of strings in a reasonable amount of time (five hours in our experiments). This phenomenon can easily be explained by the random-search-based principle utilized in evolutionary algorithms. Random application-to-machine assignments generated in the solution space resulted in too many applications mapped on a single machine or communication route eventually violating the QoS constraint.

Therefore, the Genitor-based evolutionary algorithm presented in this section was modified to search over the permutation space instead of directly over the solution space. An ordering of strings in the permutation space is translated into a mapping in the solution space by repetitively applying the Incremental Mapping Routine (IMR) described below. The IMR is designed to map a single string; different allocations are achieved when different orderings of strings are sequentially processed by the IMR.

Our choice of the Genitor-based evolutionary algorithm was based on high performance results that this algorithm evidenced in many problem domains related to resource allocation in distributed systems (e.g., [32, 97]). Furthermore, based on our previous experiments, the convergence rate of Genitor-based algorithms is usually higher than that of other modern evolutionary heuristics, e.g., Simulated Annealing [70], Ant Colony Optimization [33].

2.5.2 Incremental Mapping Routine

The allocation algorithm used in the IMR heuristic is based on a greedy mapping technique. The IMR handles one string at a time, retrieving applications in the string for mapping in a certain order, and having its resource-candidate search guided by

impact on the resource utilization. Starting from the most computationally intensive application (on average), determined in step 1 of the pseudo code shown below, the heuristic maps all the intermediate applications along the string up to the next most computationally intensive application (on average). In selecting a mapping, a parameter of interest is the maximum value of the resource utilizations (given by equations (2.2) and (2.3)) in the machine-route pair affected by an application assignment. The selection process determines a machine for mapping by finding the minimum value of this parameter, with ties broken arbitrarily. Then, the next unassigned most computationally intensive application is found, and the same mapping procedure is repeated until the allocation for a given string is completed. The IMR approach attempts to map computationally intensive applications early, but also maps their neighboring applications, so that network utilization is taken into account as the heuristic progresses.

To describe the IMR heuristic in detail some additional notation must be introduced. Let $\underline{U^{machine}[j,i,k]}$ be the utilization of machine j if application a_i^k were assigned to machine j (in addition to the applications assigned previously to this machine). Similarly, let $\underline{U^{route}[j_1,j_2,i,k]}$ be the utilization of the communication route if application a_i^k were assigned to machine j_1 and passed its output to its successor mapped on machine j_2. Let the average nominal execution time $\underline{\overline{t}_{av}^k[i]}$ for application a_i^k be given as $\overline{t}_{av}^k[i] = \frac{1}{M} \times \sum_{j=1}^{M} \overline{t}^k[i,j]$, and let the average nominal machine CPU utilization, $\underline{\overline{u}_{av}^k[i]}$ for application a_i^k be given as $\overline{u}_{av}^k[i] = \frac{1}{M} \times \sum_{j=1}^{M} \overline{u}^k[i,j]$. A detailed description of the IMR heuristic follows.

1. As a starting point, identify application $a^k_{i_{max}}$ in the given string S^k as follows:

$$i_{max} = \underset{i=1,\ldots,n_k}{\operatorname{argmax}} \left\{ \frac{\overline{t}^k_{av}[i] \times \overline{u}^k_{av}[i]}{P[k]} \right\}.$$

2. **If** $1 < \min_{j=1,\ldots,M} \{U^{machine}[j, i_{max}, k]\}$
   return mapping failed.

3. Assign application $a^k_{i_{max}}$ to the machine $m_{i_{max},k}$ found as:

$$m[i_{max}, k] = \underset{j=1,\ldots,M}{\operatorname{argmin}} \{U^{machine}[j, i_{max}, k]\}.$$

4. Initialize set $D = \{a^k_{i_{max}}\}$.

5. **While** set D does not contain all applications in the given string S^k **do**

   (a) i_{right} = max application index in D; i_{left} = min application index in D;

   (b) identify a new unassigned application $a^k_{i_{max}}$ in the given string S^k as follows:

$$i_{max} = \underset{i=1,\ldots,n_k \;\&\; a^k_i \notin D}{\operatorname{argmax}} \left\{ \frac{\overline{t}^k_{av}[i] \times \overline{u}^k_{av}[i]}{P[k]} \right\};$$

   (c) **while** $i_{max} > i_{right}$ **do**

   - $i_{right} = i_{right} + 1$;
   - **if**

$$1 < \min_{j=1,\ldots,M} \max\{U^{machine}[j, i_{right}, k] U^{route}[m[i_{right} - 1, k], j, i_{right}, k]\}$$

return mapping failed;

- assign to the machine found as follows:

$$m[i_{right}, k] = \underset{j=1,\ldots,M}{\operatorname{argmin}} \left[\max\{U^{machine}[j, i_{right}, k], U^{route}[m[i_{right} - 1, k], j, i_{right}, k]\}\right];$$

- insert application $a^k_{i_{right}}$ in set D;

(d) **while** $i_{max} < i_{left}$ **do**

- $i_{left} = i_{left} - 1$;
- **if**

$$1 < \min_{j=1,\ldots,M} \left[\max\{U^{machine}[j, i_{left}, k], U^{route}[j, m[i_{left} + 1, k], i_{left}, k]\}\right]$$

return mapping failed;

- assign $a^k_{i_{left}}$ to the machine $m[i_{left}, k]$ found as follows:

$$m[i_{left}, k] = \underset{j=1,\ldots,M}{\operatorname{argmin}} \left[\max\{U^{machine}[j, i_{left}, k], U^{route}[j, m[i_{left} + 1, k], i_{left}, k]\}\right];$$

insert application $a^k_{i_{left}}$ in set D.

2.5.3 Permutation Space Genitor-Based Heuristic

The permutation space Genitor-based heuristic, PSG, was developed by combining the IMR heuristic with concepts from the Genitor approach. Genitor is an evolutionary steady-state genetic search algorithm that has been shown to work well for several problem domains (e.g., [17,54,102]). Designed for a given resource allocation

problem, each chromosome in the heuristic represents an ordered list of strings in the permutation space. Genitor-specific operators, such as selection, crossover, and mutation, are applied in that space. Chromosomes differ in their list orders, which results in different mappings in the solution space obtained via "projecting" a chromosome to the solution space by applying the IMR.

If the IMR mapping fails for a string due to a utilization violation, then the string is skipped, and IMR proceeds with the next string. The two-component performance metric is used to measure the fitness of each chromosome. Recall from Section 2.4 that the primary component of the performance metric indicates total worth of the strings allocated in the system while the secondary component indicates system slackness.

The PSG heuristic was implemented as follows. First, an initial population is generated randomly by reordering the initial set of strings. A population size of 250 chromosomes was determined experimentally for a given setup; any further increase in the size of the population does not improve the performance of the heuristic. After a mapping involving the IMR procedure, the entire set of chromosomes is sorted (ranked) by their fitness (system slackness for the complete allocation scenario; total worth for the partial allocation scenario with ties broken by system slackness). Next, a special function (described later) is used to select two chromosomes to act as parents. These two parents perform a crossover operation, and two offspring are generated. Each offspring is then inserted in the ordered population, and the worst two chromosomes are dropped.

In the crossover step, for the selected pair of parent chromosomes a random cut-off point is generated that divides the chromosomes into top and bottom parts. Next, the strings in each of the top parts are reordered. The new ordering of the strings in one top part corresponds to the relative positions of its strings in the other parent

chromosome in the pair. It is important to note the choice of the top parts of the parent chromosomes for reordering. This allows the offspring to differ from their parents in the case of a partial resource allocation.

After each crossover, the same special function (described below) is applied to select a chromosome for mutation. The mutation operator generates a single offspring by perturbing the original chromosome order via swapping two randomly chosen application strings. The resultant offspring is considered for inclusion in the population in the same fashion as an offspring generated by crossover.

The special function for selecting parent chromosome(s) is a bias function, used to provide a specific selective pressure [102]. For example a bias of 1.5 implies that the top-ranked chromosome in the population is 1.5 times more likely to be selected for a crossover or mutation than the median chromosome. Experimentally, by varying the bias values across the range [1, 2] in steps of 0.1, the best bias for this system was found to be 1.6.

As the PSG runs, the crossover operator will be iteratively repeated followed by the mutation operator until one of the stopping conditions is reached: (1) 120 minutes to execute, (2) 2000 iterations without a change in the best chromosome, or (3) either the mutation or the crossover operator failed to produce a never before examined chromosome within 10 minutes.

The developed PSG heuristic was used in the conducted experiments in the following ways:

- In each run of the complete resource allocation scenario, the PSG heuristic was applied to compute the complete initial allocation while maximizing the secondary component of the objective metric, i.e., system slackness. This baseline solution was then used in the follow-up Branch-and-Bound algorithm (described

in Section 2.7) to reduce the search space.

- In the partial resource allocation scenario, PSG was used to find a subset of mapped strings that results in the maximized total worth, using system slackness to break ties. The determined subset was then passed to the Branch-and-Bound heuristic, which aimed to improve an allocation for the subset with respect to system slackness.

2.6 Integer Linear Programming Formulation

2.6.1 Overview

An Integer Linear Programming (ILP) form and a corresponding Linear Programming (LP) form [24, 73] are derived in this section for each allocation scenario. The ILP form fully describes the optimization problem considered in each scenario while its relaxation into the LP form: (1) provides the initial upper bound on the performance metric, and (2) establishes a basis for a node selection in the Branch-and-Bound algorithm presented in the following section.

Let $\boldsymbol{c} \in \mathbb{R}^{\alpha}$ and $\boldsymbol{b} \in \mathbb{R}^{\beta}$ be real vectors,and $\boldsymbol{\Upsilon} \in \mathbb{R}^{\beta \times \alpha}$ be a real matrix. If $\boldsymbol{h}$ is a vector comprised of α decision variables [20] then the canonical ILP formulation is written as:

$$\text{maximize} \quad Z_{ILP} = \boldsymbol{c}^T \times \boldsymbol{h}; \qquad \text{subject to (I) } \boldsymbol{\Upsilon} \times \boldsymbol{h} \leq \boldsymbol{b}, \quad \text{(II) } \boldsymbol{h} \quad \text{are integers} \tag{2.5}$$

Constraint (II) makes the ILP problem NP-complete [71]. If this constraint is ignored, i.e., $\boldsymbol{h} \in \mathbb{R}^{\alpha}$ then the ILP form is relaxed into an LP form. The global optimal solution for the LP form, which is the upper bound (UB) for the ILP form, can be found in

polynomial time, e.g., by applying one of the interior-points methods [44].

Let the binary decision variable $\underline{x[i,k,j]}$ be equal to one if application a_i^k is assigned to machine j and equal to zero if a_i^k is not assigned to machine j. Similarly, let $\underline{y[i,k,j_1,j_2]}$ be equal to one if the output generated by a_i^k is transferred over the communication route from machine j_1 to machine j_2 and zero if it is not transferred over that communication route. Due to the new variables, equation (2.2) for machine utilization and equation (2.3) for communication route utilization need to be restated:

$$U^{machine}[j] = \sum_{k=1}^{A} \sum_{i=1}^{n_k} \left(\frac{\bar{t}^k[i,j]}{P[k]} \times \overline{u}^k[i,j] \times x[i,k,j] \right) \tag{2.6}$$

$$U^{route}[j_1,j_2] = b[j_1,j_2] \times \sum_{k=1}^{A} \sum_{i=1}^{n_k-1} \left(\frac{O^k[i]}{P[k]} \times y[i,k,j_1,j_2] \right) \tag{2.7}$$

2.6.2 Complete Allocation Scenario

In the complete resource allocation scenario, the objective is to *maximize* system slackness Λ given by (2.4), because all mappings would have the same total worth. Thus, the objective function for the ILP form is formally stated as:

$$\text{maximize} \quad \Lambda = \min\left(\{1 - U^{machine}[j] : j \in M\} \cup \{1 - U^{route}[j_1,j_2] : j_1, j_2 \in M\}\right) \tag{2.8}$$

Suppose that $\underline{Q}$ represents the total number of strings considered for mapping in the system. The objective function is subject to set of conditions (a)-(f), explained in detail below:

$$x[i,k,j] \in \{0,1\} \qquad \text{for } 1 \le i \le n_k, \quad 1 \le k \le Q, \quad 1 \le j \le M; \tag{a}$$

$$\sum_{j=1}^{M} x[i,k,j] = 1 \qquad \text{for } 1 \le i \le n_k, \quad 1 \le k \le Q; \tag{b}$$

$$x[i,k,j_1] = \sum_{j_2=1}^{M} y[i,k,j_1,j_2] \qquad \text{for } 1 \le i \le n_k - 1, \quad 1 \le k \le Q, \quad 1 \le j_1 \le M; \tag{c}$$

$$x[i,k,j_2] = \sum_{j_1=1}^{M} y[i,k,j_1,j_2] \qquad \text{for } 2 \le i \le n_k, \quad 1 \le k \le Q, \quad 1 \le j_2 \le M; \tag{d}$$

$$U^{machine}[j] \le 1 \qquad \text{for } 1 \le j \le M; \tag{e}$$

$$U^{route}[j_1,j_2] \le 1 \qquad \text{for } 1 \le j_1, j_2 \le M. \tag{f}$$

Condition (a) explicitly restricts decision variables $x[i,k,j]$ to integer binaries {0,1} corresponding to the "assigned/not assigned" allocation choice for application a_i^k on machine j. Condition (b) forces each application to be mapped to the system. Conditions (c) and (d) link the communication route assignment of output $O^k[i]$ generated by application a_i^k to the allocation of applications a_i^k and a_{i+1}^k. The enforcement of utilization feasibility in the system is represented by the remaining two conditions (e) and (f). The objective function (8) and conditions (e) and (f) are based on equations (6) and (7). The binary restriction on decision variables $y[i,k,j_1,j_2]$ is imposed implicitly by conditions (a), (c), and (d).

The objective function (2.8) and the set of conditions (a) (f) formulate an optimization problem in the complete allocation scenario in the ILP form. The LP form

that provides a UB on system slackness follows from the ILP form as the decision variables in condition (a) are relaxed to real numbers, i.e., for $1 \leq i \leq n_k, \quad 1 \leq k \leq Q, \quad 1 \leq j \leq M, \quad 0 \leq x[i,k,j] \leq 1$. This implies that each $y[i,k,j_1,j_2]$ is also a real number.

2.6.3 Partial Allocation Scenario

In the partial resource allocation scenario the primary objective is to maximize the total worth of the strings deployed in the system, as defined in Section 2.4. This transforms into the formal representation of an objective function in the ILP form:

$$\text{maximize} \sum_{k=1}^{Q} \sum_{i=1}^{n_k} \left(W[k] \times \sum_{j=1}^{M} x[i,k,j] \right) \tag{2.9}$$

The objective function is subject to conditions (a)–(f), where condition (b) needs to be restated due to the limited computation or communication capacity of the resources available in the partial allocation scenario:

$$\sum_{j=1}^{M} x[i,k,j] \in \{0,1\} \quad \text{for } 1 \leq i \leq n_k, \quad 1 \leq k \leq Q. \qquad \text{(b')}$$

The modified condition (b'), along with conditions (a), (c), and (d), requires each of Q strings to be either completely mapped or not mapped at all, precluding cases where the number of mapped applications in the string is less than n_k. An LP form that provides an upper bound on total worth in the partial allocation scenario is obtained from the derived ILP form when conditions (a) and (b') are relaxed to real numbers confined to the interval [0,1].

The LP forms presented above result in the initial upper bounds on system slackness and total worth in the complete and partial allocation scenarios, respectively. However, tighter upper bounds were achieved iteratively in both scenarios by applying the developed Branch-and-Bound algorithms described in the next section.

2.7 Branch-and-Bound Heuristics

Due to the NP-complete nature of the ILP problem, in general its global optimal solution cannot be found in polynomial time except for some special cases described in the literature (e.g., [19, 73, 103]). In the complete allocation scenario, the Branch-and-Bound (B&B) algorithm, presented in this section, was designed to improve a suboptimal solution produced by the PSG algorithm and to tighten the initial UB on system slackness. In the partial allocation scenario, the B&B algorithm was developed to tighten the UB on total worth.

2.7.1 Complete Allocation Scenario

The proposed B&B algorithm is a tree search beginning at a root node that is a null solution. In the entire tree, interior nodes represent intermediate solutions (a subset of applications are assigned to machines), and leaf nodes represent final solutions (all applications are assigned to machines). The intermediate solution of a child node has one more application mapped than its parent node. Call this additional application a. Each parent node expands into M children, one for each possible mapping of a. Nodes are said to be open until they are expanded, whereupon they become closed.

The intermediate solution at each node is characterized by a value of the secondary component of the objective function (system slackness) found by solving the LP form,

derived in Section 2.6, for this scenario. When solving this LP form, the decision variables that correspond to applications already mapped are set to binary integers {0,1} according to application assignments; other decision variables are relaxed to real numbers. The LP solutions at nodes are used as bounds to curtail the search. Specifically, a node is pruned (closed) if one of the following two conditions holds: (I) no solution can be found for the LP form at a given node, i.e., no solution can be found that satisfies the set of constraints (a)-(g); (II) the value of the objective function found for the LP form is not greater than the highest value of the objective function among the known final solutions.

As it follows from condition (II), a known high-quality final solution helps to avoid the explicit examination of many early nodes in the tree and, thus, significantly narrows the search space. In the proposed B&B algorithm, the baseline solution generated by the developed PSG heuristic is used for pruning until B&B determines a better final solution from a leaf node. Although pruning helps to limit the search space, the problem of finding the global optimal ILP solution in the complete allocation scenario remains quite time-consuming due to the large solution space comprised of numerous application-to-machine assignment combinations. Therefore, the total execution time for the B&B heuristic was limited to five hours.

The B&B algorithm can be summarized by the following procedure. Starting from the root node, the B&B heuristic iteratively attempts to reach the bottom level of the tree by selecting a new parent from among the open children resultant from the previous expansion. The child selected is the one with the maximum objective function value. Such a node expansion method is referred in the literature as a depth-first search [73]. When the bottom level of the tree is reached, M final solutions are generated. If the best of these final solutions has an objective function value higher

than that used for pruning before, this new final solution is now the overall best found so far, and will be used for future pruning. Furthermore, this final solution is applied to evaluate all open intermediate nodes currently included in the tree to close the nodes that satisfy condition (II). If none of the nodes considered for picking a parent node is open or the bottom level of the tree is reached, a new startup node is selected in the tree to continue the B&B search process. The startup node selection, called backtracking [73], is based on the highest objective function value among the LP solutions associated with all *open* interior nodes currently included in the tree. It is important to note that the LP value of a new startup node is a new UB for the considered allocation problem. Thus, every time a new startup node is selected, the UB becomes tighter if the new node's LP solution differs from that of the previous startup node. The described B&B search process continues until: (1) the execution time limit (five hours) is reached; (2) all the nodes in the tree are pruned except for a single leaf node, i.e., the global optimal solution is found. Typically, due to the NP-complete nature of the algorithm, stopping condition (2) is unlikely to occur when the solution space of the problem is relatively large.

An important issue for the B&B algorithm is the order in which applications are considered in the node expansion process. In the conducted experiments, three different orderings of applications were tested to identify the one resulting in the best performance. An arbitrary ordering implies that applications from Q strings are randomly shuffled. As opposed to such a random-based approach, a max-first ordering contains applications ranked in descending order of their average load estimate $AL^k[i]$, which is associated with each application i in string $S^k : 1 \leq k \leq Q$, and defined as:

$$AL^k[i] = \frac{1}{P[k]} \times \left[\frac{1}{M} \times \sum_{j=1}^{M} \overline{u}^k[i,j] \times \overline{t}^k[i,j] + b_{av} \times O^k[i] \right].$$

Applications in a min-first ordering are ranked in ascending order of their average load estimate.

2.7.2 Partial Allocation Scenario

Three major goals were addressed in the partial allocation scenario by applying the Branch-and-Bound technique based on LP formulations derived in Section 2.6: (1) finding a tighter bound than the initial UB on total worth achievable in the system for a given set of Q strings; (2) finding a tighter bound than the initial UB on system slackness achievable for the subset of A strings, $A \leq Q$; (3) making an attempt to find a better allocation for the subset of A strings to maximize system slackness. The subset of A strings, referred to in (2) and (3), is essentially the subset of mapped strings in the best chromosome produced as the PSG heuristic terminates, characterized by the highest known total worth value. Goals (2) and (3) with respect to the subset of A strings are identical to the goals in the complete allocation scenario with respect to the set of Q strings. As such, the B&B algorithm designed for the complete allocation scenario can be applied to address (2) and (3) when the set of Q strings is replaced with the subset of A strings in the corresponding LP formulation.

For goal (1), a tighter bound than the initial UB on total worth can be found by applying another B&B algorithm to find a solution for the ILP form given by objective function (2.9) and conditions (a), (b′), (c)–(f), where condition (a) is relaxed to allow for real numbers confined to the interval [0,1]. The node expansion and backtracking mechanisms in this B&B remain identical to those designed for the complete allocation scenario. In contrast, the tree structure considered here is different—nodes are associated with strings as opposed to the complete allocation scenario where nodes

are associated with applications. Each parent node generates two children when expanded where the children represent the cases when a new string is either loaded or not loaded to the system. The term "loaded" is used here as opposed to "mapped" to emphasize that no actual application assignments are produced due to relaxed condition (a). Call a string from the set of Q strings processed when the two nodes corresponding to the loaded/not loaded decisions made for that string are included in the tree. In the LP form for a given interior node, the sum in condition (b') is set to 0 or 1 for the processed strings, and relaxed to real numbers confined to the interval [0,1] for the others. The total worth value found by the PSG heuristic is used for the initial node pruning until B&B finds a better final solution (all strings are processed). In contrast to a large solution space in the complete allocation scenario, where the number of leaf nodes could reach $M^{\sum_{k=1}^{Q} n_k}$, the solution space that needs to be explored for the considered ILP form is significantly smaller, comprised of at most 2^Q leaf nodes. Thus, the B&B heuristic is able to converge to the global optimal solution in a reasonable amount of time for a given simulation setup (about one hour). Recall that the discussed ILP form does not represent the actual allocation problem due to relaxed condition (a). The order in which Q strings are processed in the tree search affects the convergence time because early improvement of the best known final solution helps to curtail the search space and avoid redundant computations. Three different orderings of strings were considered in the experiments. An arbitrary ordering is based on a random arrangement of Q strings. A max-first ordering contains strings ranked in descending order of their average worth per average load estimate

$AWAL[k]$, defined for each string as:

$$AWAL[k] = \frac{W[k]}{\sum_{i=1}^{n_k} AL^k[i]}.$$

Strings in a min-first ordering are ranked in ascending order of their $AWAL[k]$.

2.8 Simulation Experiments and Results

2.8.1 Simulation Setup

The purpose of the simulation was to evaluate the performance of the developed mapping heuristics in two different workload scenarios. For each scenario, the hardware part of the intended system was composed of a heterogeneous suite of eight machines. The bandwidth of each inter-machine communication route was determined by sampling a uniform distribution on the interval between 1 and 10 Mb/sec. All intra-machine communication routes were assumed to have infinite bandwidth, i.e., $b[j, j] = 0$. In addition, the time-of-flight, i.e., time needed for the first transmitted bit of data to reach the destination [46], was assumed to be negligible on each communication route. For all experiments, it also was assumed that an application could execute on any machine, and its output could be transferred over any communication route. The two workload scenarios were distinguished by a different number of strings considered for mapping and different ranges for the periods of the strings.

Recall that a partial allocation scenario occurs in the oversubscribed system when not all the strings in a given set can be successfully allocated because some hardware component in the system would exceed its 100% utilization limit. To model this situation, a set of 75 strings was generated and the string periods were set to be tight,

as explained below. For the complete allocation scenario 45 strings were created with more relaxed period values.

The developed heuristics were tested for operation with strings composed of a different number of applications determined randomly within the range from 1 to 10. The nominal execution time and nominal machine CPU utilization requirement associated with each application in the string were set by sampling a uniform distribution in the intervals between 1 and 10 seconds, and between 0.1 and 1, respectively. In the same fashion, the size of the output generated by each application in the string was chosen in the interval from 10 to 100 Kbytes.

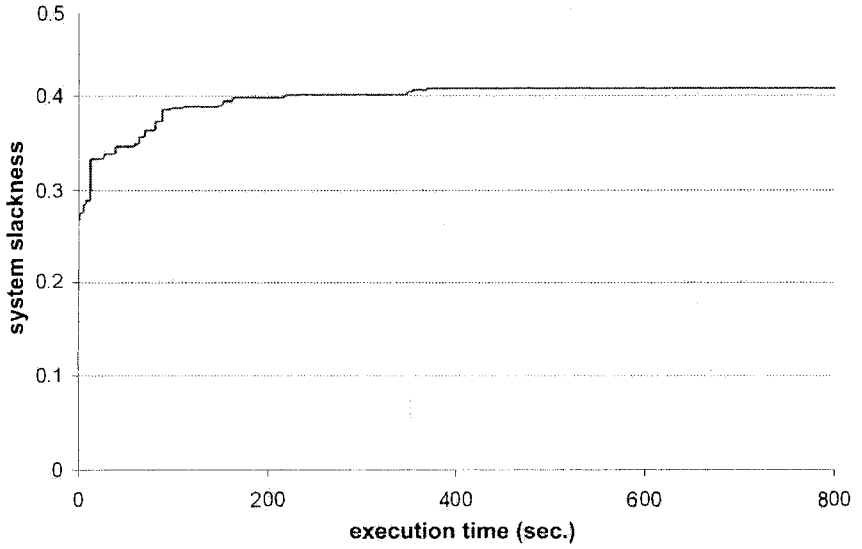
In addition to average nominal execution time $\bar{t}^k_{av}[i]$, introduced in Section 2.5, let the average time to transmit one bit of data in the system, $\underline{b_{av}}$, be calculated as the average across all possible communication routes in the system: $b_{av} = \frac{1}{M^2} \sum_{j_1=1}^{M} \sum_{j_2=1}^{M} b[j_1, j_2]$. The random variable $\underline{\mu}$ with a uniform distribution in a particular range that was inserted to control the tightness of period $P[k]$ associated with each string. All these new variables were combined in the following way to set a period of string S^k: $P[k] = \mu \times \max\{\bar{t}^k_{av}[i] : 1 \leq i \leq n_k, \quad b_{av} \times O^k[z] : 1 \leq z \leq n_k - 1\}$. The range for the random variable μ for the complete allocation scenario was set to [3, 4.5], and for the partial allocation scenario was set to [2, 3].

An interactive software framework has been developed for this study that allows for simulation, testing, and result visualization of the designed mapping techniques. The optimization package Lingo 9.0 was employed to compute LP solutions in the B&B algorithms. For each scenario, 50 simulation runs were performed which allows for a 95% confidence interval [29] computation.

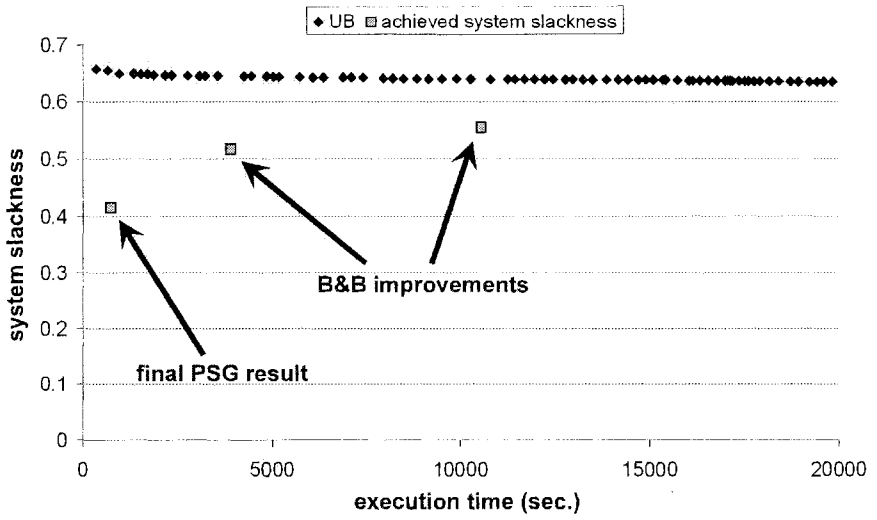
2.8.2 Experimental Results

Fig. 2.3(a) demonstrates how the system slackness was gradually improved by the PSG heuristic over time for a typical run in the complete allocation scenario. In 87% of the runs PSG was able to generate unique offspring while performing mutation and crossover operations, and the heuristic was terminated when the second stopping criterion, i.e., 2000 iterations without a change in the best chromosome, was reached. Additional experiments revealed that an increase in the maximum number of iterations without a change in the best chromosome does not affect the performance of the heuristic. In the remaining 13% of the runs, at some point in time PSG failed to produce a new unique chromosome within 10 minutes and was terminated. A typical run of the PSG heuristic lasted for less than 16 minutes. Fig. 2.3(b) shows an example where the final PSG result passed to the follow-up B&B algorithm was improved twice. In addition to that, the UB on system slackness was tightened by B&B as the algorithm progressed. A tight UB is very important to evaluate the quality of the final result—in practice such an evaluation can be used to determine when the algorithm needs to be stopped.

The performance results of 50 runs against the secondary objective metric component are shown for the complete allocation scenario in Fig. 2.4. In 34% of the cases, the B&B algorithm was able to improve the PSG results, i.e., to find a resource allocation with a higher value of system slackness. Fig. 2.4 depicts the best performance achieved when the max-first ordering was used to rank applications while constructing a tree in the B&B algorithm. Table 2.3 compares the efficiency of the max-first, min-first, and arbitrary orderings for the system slackness improvement and UB tightening. The results were averaged across 50 runs for each of the orderings. The highest efficiency exhibited by the max-first ordering is based on the fact



(a)



(b)

Figure 2.3: System slackness in the complete allocation scenario achieved over time in a single run by applying the two-stage resource allocation method: (a) progress of the PSG heuristic; (b) the final PSG result passed to the follow-up B&B algorithm was improved twice. The UB on system slackness was tightened by B&B as the algorithm progressed.

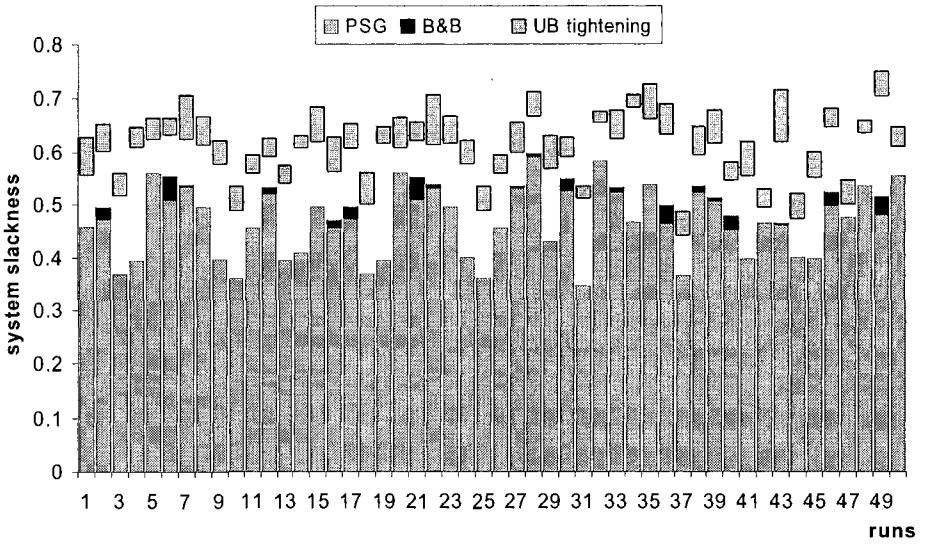
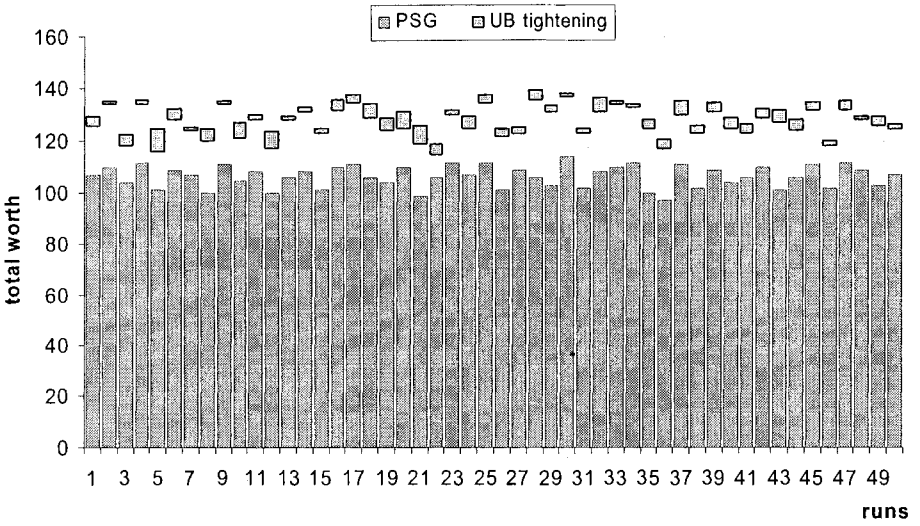
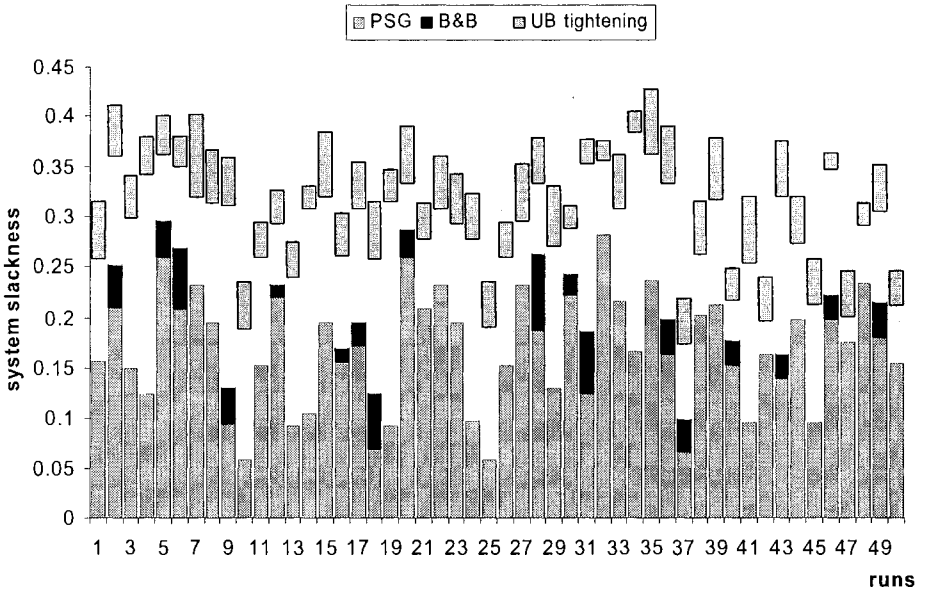


Figure 2.4: The system slackness achieved by PSG with a follow-up improvement provided by B&B in the complete allocation scenario. The result per each run is plotted along with the UB tightened by B&B.

that allocations for applications were resolved in the tree in descending order of their average load estimate. This ordering corresponds to the well-known bin-packing principle [68] implying that allocation is first resolved for the workload components with high resource consumption requirements. In contrast, the performance results of UB tightening on system slackness revealed the B&B algorithm to be relatively insensitive to application orderings. Averaged across 50 runs, the achieved system slackness including any B&B improvements was 0.47 for the complete allocation scenario, and 0.11 for the partial allocation scenario; the achieved system slackness per run normalized against the corresponding UB was 81% for the complete allocation scenario, and 83% for the partial allocation scenario. The much lower absolute system slackness in the partial allocation scenario must be expected: the system is "packed" with applications to the point where the remaining slack is not adequate to accommodate an extra string.



(a)



(b)

Figure 2.5: Performance in the partial allocation scenario for each run: (a) total worth achieved by PSG and UB tightened with B&B; (b) system slackness achieved by PSG with a follow-up improvement provided by B&B and UB tightened by B&B.

Table 2.3: Performance of the B&B algorithm improving system slackness averaged across 50 runs in the complete and partial allocation scenarios.

scenario	ordering	system slackness improvement			UB	
		successful runs (%)	improvement over PSG (%)	95% conf. interval	aver.	95% conf. interval
complete allocation	max-first	34	4.65	[2.92, 6.38]	0.59	[0.55, 0.63]
	min-first	28	2.87	[0.52, 5.22]	0.59	[0.57, 0.61]
	arbitrary	10	2.54	[1.32, 3.76]	0.59	[0.57, 0.61]
partial allocation	max-first	40	6.46	[4.12, 7.81]	0.32	[0.26, 0.38]
	min-first	30	3.25	[2.36, 4.14]	0.32	[0.25, 0.39]
	arbitrary	8	4.42	[1.78, 7.84]	0.34	[0.27, 0.41]

For the partial allocation scenario, the experimental results obtained with the PSG heuristic for the primary component of the performance metric (total worth) are shown in Fig 2.5(a). The performance for each run is plotted along with the corresponding UB tightened by utilizing the developed B&B algorithm. The convergence to a tighter UB was reached in 4.7 minutes on average across 50 runs when strings were arranged in the max-first order, 6.3 minutes when strings were arranged in the min-first order, and 7.4 minutes when strings were arranged in an arbitrary order. The PSG heuristic performed well in this scenario achieving mappings that averaged above 80% of the upper bound.

Fig. 2.5(b) illustrates the performance of the PSG and the follow-up max-first variant of the B&B heuristics maximizing the secondary component of the objective metric, i.e., system slackness. As Table 1 shows, compared to the complete allocation scenario, all three variants of the B&B heuristic succeeded in system slackness improvement over the PSG results in approximately the same number of runs, but their relative improvement on system slackness was higher.

2.9 Related Work

A number of papers in the literature have studied the issue of finding an initial resource allocation that is robust against unpredictable workload increases (e.g., [5, 6, 23, 25, 31, 39, 43, 48]). These studies are compared below.

The nature of the problem described in [5] is similar to the presented problem in this chapter. Periodically running applications are organized in sequential strings, which are subject to the imposed end-to-end latency and throughput constraints. In that study it is assumed that the computation time of an application sharing a given machine with $N - 1$ other applications was N times its nominal execution time. This results in conservative execution time estimates in a shared environment. Furthermore, there is no notion of nominal utilization—i.e., it is assumed that all applications utilize 100% of the CPU when executing. Our research does not make such assumptions about execution time and CPU utilization; therefore, the approach taken is quite different from that in [5].

Slack-based techniques explored in this work approach robust resource allocation by increasing the amount of unused computation or communication capacity across all hardware resources in the system. A similar performance metric was applied in [25] and [43] to achieve robust schedules in job-shop and real-time environments, respectively. Specifically, an attempt in those works was made to provide each task with extra time (defined as slack) to execute so that some level of uncertainty can be tolerated without having to reallocate.

In [6], it was demonstrated that when application execution parameters are known as a function of workload then a measure of robustness better than system slackness could be used. However, in the given shipboard environment, such a function is unknown, and therefore the system slackness is an appropriate measure to use.

The research in [23] considers a single-machine scheduling environment where the processing times of individual jobs are uncertain. The system performance is measured by the total flow time (i.e., the sum of *completion* times of all jobs). Given the probabilistic information about the processing time for each job, the authors determine the normal distribution that approximates the flow time associated with a given schedule. A given schedule's robustness is then given by one minus the risk of achieving substandard flow time performance. The risk value is calculated by using the approximate distribution of flow time. It is important to note that, in contrast to [23], the workload increases are expected in the ARMS environment but not specified stochastically. If this information was known, the accuracy of a robustness metric could be improved by using techniques similar to those in [23].

Our combination of evolutionary algorithms with the IMR heuristic is conceptually similar to [31] and [101]. For example, in [31] the goal is to minimize a weighted combination of the cost of the system and the execution time of a set of tasks. A genetic algorithm manipulates a set of chromosomes, where each chromosome is composed of a subset of resources available in the system, and an ordering of tasks. A separate greedy heuristic operates on each chromosome to derive a mapping and the associated execution time for the set of tasks.

As opposed to heuristic scheduling algorithms finding approximate (or suboptimal) solutions, exact algorithms for finding optimal solutions are based on Integer Linear Programming. Although solving an ILP formulation is NP-hard, significant progress has been made in the development of efficient ILP algorithms. For example, ILP-schedulers for VLSI architectural synthesis, such as OASIC [39] and ALPS [48], have produced better designs than heuristic algorithms for medium-sized problems in comparable time. However, with an increase in the problem scale the performance of

ILP-schedulers degrades significantly while heuristic approaches are still able to produce high-quality solutions in a reasonable amount of time. In our work, a specially designed first-stage evolutionary heuristic was utilized to find a high-quality baseline solution that was used efficiently in the second-stage B&B algorithm to narrow the search. As a result, the baseline solutions were improved in at least 34% of cases for the considered resource allocation problems.

2.10 Summary

This chapter presents methods for efficiently and robustly managing both computation and communication resources in the intended distributed system. The system is expected to operate in an unpredictable environment where the workload might increase, possibly invalidating a resource allocation that was based on the initial workload estimate. The focus in the design of the allocation heuristics was to achieve the highest level of total worth of the strings deployed in the system while maximizing system slackness at that level. System slackness is a measure that quantitatively reflects the system's potential to absorb unpredictable increases in input workload.

Formed by combining the efficient evolutionary Genitor-based search methods with a specially designed string allocation routine IMR, the PSG heuristic was used to generate baseline solutions in the complete and partial allocation scenarios. Further resource allocation improvement with respect to system slackness, and iterative UB tightening for both objective metric components, were based on the developed B&B algorithms. To establish the foundation for these algorithms, the considered resource allocation problems were formulated in the ILP form. Due to the high-quality of the

baseline solutions provided by the PSG, the search spaces explored by the B&B algorithms were significantly reduced. As a result, the B&B algorithm succeeded in 34% of the experiment runs with 4.65% improvement over the PSG results in the complete allocation scenario, and in 40% of the experiment runs with 6.46% improvement over the PSG results in the partial allocation scenario. By demonstrating a performance ranging from 81% to 83% of the upper bound, the proposed combinatorial mapping approach indicates a significant potential to produce effective resource allocations in an environment associated with unpredictable workload increases.

Chapter 3

Resource Allocation in a Cluster Based Imaging System *

3.1 Overview

Recently there has been an increased demand for imaging systems in support of high-speed digital printing. The required increase in performance in support of such systems can be accomplished through an effective parallel execution of image processing applications in a distributed cluster computing environment. The output of the system must be presented to a raster based display at regular intervals, effectively establishing a hard deadline for the production of each output image. Failure to complete a rasterization task before its deadline will result in an interruption of service that is unacceptable. The goal of this research was to derive a metric for measuring robustness in this environment and to design a resource allocation heuristic capable of completing each rasterization task before its assigned deadline, thus, preventing any

*This entire chapter was done jointly with James Smith and appears in the following paper: [94]. A part of this material was filed as a patent to the U.S. Patent Office [90].

service interruptions. We present a mathematical model of such a cluster based raster imaging system, derive a robustness metric for evaluating heuristics in this environment, and demonstrate using the metric to make resource allocation decisions. The heuristics are evaluated within a simulation of the studied raster imaging system. We clearly demonstrate the effectiveness of the heuristics by comparing their results with the results of a resource allocation heuristic commonly used in this type of system.

3.2 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 acheived 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 [42]. 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 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 subtantially 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 [65]. The general problem of assigning tasks to workstations in a dynamic environment has been shown to be NP-complete (e.g., [35, 49]). Consequently, the design of heuristics for dynamic resource allocation is an active area of research [10, 37, 65, 69, 93, 105].

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 3.4. Section 3.5 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 3.6. This performance metric motivated the derivation of the robustness metric in Section 3.7. The details of the simulation setup are described in Section 3.8 and the results of the heuristics are presented in Section 3.9. A sampling of related work is in Section 3.10 and Section 3.11 concludes the paper.

3.3 System Model

Figure 3.1 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 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

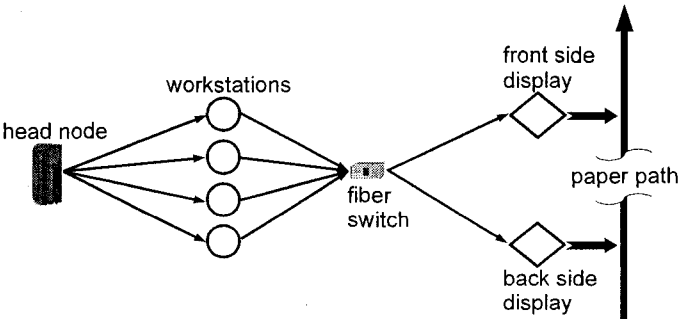


Figure 3.1: A conceptual model of a high performance, cluster-based imaging system.

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 $\underline{M}$ 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.

3.4 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 ($t_{display}$ is a fixed prespecified value). 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 3.1). 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.

$$t_d[S_k] = \begin{cases} t_0 + t_{display}\left(\frac{S_k-1}{2}\right) - t_{tran}^{bitmap} & \text{if } S_k \text{ is odd;} \\ t_0 + t_{display}\left(\frac{S_k}{2}\right) - t_{tran}^{bitmap} & \text{if } S_k \text{ is even} \end{cases} \tag{3.1}$$

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 3.1 can be replaced by $\text{num}(BQ_i^j)$. Note that S_k and BQ_i^j represent the same physical sheetside 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 $\underline{t_{start}[BQ_i^j]}$ be the earliest possible rasterization start time for sheetside BQ_i^j, let $\underline{ERT[BQ_i^j]}$ be the estimated rasterization time for sheetside BQ_i^j, and let $\underline{t_{comp}[BQ_i^j]}$ 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:

$$t_{comp}[BQ_i^j] = t_{start}[BQ_i^j] + ERT[BQ_i^j]. \tag{3.2}$$

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 [41, 53, 60, 92, 106]. 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 accommodate 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 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:

$$AC_{in}^j = CAP_{in}^j - \sum_{\forall S_i \in \mathcal{K}} \text{size}(S_i). \tag{3.3}$$

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

```
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
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
```

Figure 3.2: Pseudo-code for determining the earliest estimated departure time for sheetside S_k assigned to workstation j.

S_{k-1}, from HNIQ to workstation x. If size$(S_k) \leq AC_{in}^j$ and $|\mathcal{K}| < Q$, S_k can depart at time:

$$t_{dept}^j[S_k] = t_{dept}^x[S_{k-1}] + t_{tran}^{sdf}[S_{k-1}]. \tag{3.4}$$

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 3.2 presents a concise pseudo-code form for determining the earliest estimated depature 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 $\underline{CAP^j_{out}}$. 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^j_i, denoted $\underline{\Delta_{out}[BQ^j_i]}$, as the time that BQ^j_i 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 $\Delta_{out}[BQ^j_i]$, 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., $\Delta_{out}[BQ^j_i] = 0$. 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^j_i completes there will be at least one free slot in the output buffer of workstation j, i.e., at least sheetside BQ^j_{i-N} has left the output buffer, then $\Delta_{out}[BQ^j_i] = 0$. In the final case, if the output buffer of workstation j is full when sheetside BQ^j_{i-1} completes, then BQ^j_i must wait for an opening in the output buffer before its processing can begin. Therefore, sheetside BQ^j_i 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^j_i to begin processing can be described succinctly as follows.

$$\text{Case 1: if } i < N, \Delta_{out}[BQ^j_i] = 0 \tag{3.5}$$

$$\text{Case 2: if } t_d[BQ_{i-N}] + t^{bitmap}_{tran} \leq t_{comp}[BQ^j_{i-1}], \Delta_{out}[BQ^j_i] = 0 \tag{3.6}$$

$$\text{Case 3: otherwise,} \Delta_{out}[BQ^j_i] = t_d[BQ^j_{i-N}] + t^{bitmap}_{tran} - t_{comp}[BQ^j_{i-1}] \tag{3.7}$$

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:

$$t_{start}[BQ_i^j] = t_{comp}[BQ_{i-1}^j] + \Delta_{out}[BQ_i^j]. \tag{3.8}$$

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.:

$$t_{start}[BQ_i^j] = t_{dept}[BQ_i^j] + t_{tran}^{sdf}[BQ_i^j]. \tag{3.9}$$

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:

$$t_{start}[BQ_i^j] = \max\{\left(t_{comp}[BQ_{i-1}^j] + \Delta_{out}[BQ_i^j]\right), \left(t_{dept}[BQ_i^j] + t_{tran}^{sdf}[BQ_i^j]\right)\}. \tag{3.10}$$

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:

$$t_{comp}[BQ_1^j] = ERT[BQ_1^j] + t_{dept}[BQ_1^j] + t_{tran}^{sdf}[BQ_1^j]. \tag{3.11}$$

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.

3.5 Minimum Rasterization Completion Time Heuristic (MRCT)

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.

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:

$$INVT_j = EEFT_j + \left(t^x_{comp}[S_k] - t^j_{comp}[S_k]\right). \tag{3.12}$$

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.

3.6 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.

3.7 Quantifying Robustness

3.7.1 Overall Robustness Metric

To derive a robustness metric suitable for this environment, we follow the FePIA procedure presented in [6]. 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 [6], 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 3.7.2, we have chosen to approximate this surface by a hyperplane. Using the equation for the distance from a point to a hyperplane [6] 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:

$$r_\mu\left(BQ_i^j, t\right) = \frac{t_d[BQ_i^j] - t_{comp}[BQ_i^j]}{\sqrt{\mathcal{B}_i^j}}. \tag{3.13}$$

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:

$$\rho_\mu^j(t) = \min \forall i \in \mathcal{B}_i^j \left\{r_\mu\left(BQ_i^j, t\right)\right\}. \tag{3.14}$$

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,

$$\rho_\mu(t) = \min \forall j \left\{ \rho_\mu^j(t) \right\}. \tag{3.15}$$

Because any service interuption 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:

$$\rho_\mu = \min \forall t \left\{ \rho_\mu(t) \right\} \tag{3.16}$$

3.7.2 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 (3.13) 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.

3.8 Simulation Setup

To evaluate our heuristics, we created a simulation model of a real printing system using the Opnet simulation environment [74]. 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 [98]. 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.

3.9 Simulation Results

Figure 3.3 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 3.6, 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 3.3, 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 soley attributed to the use of robustness in the heuristic in place of bitmap lifetime.

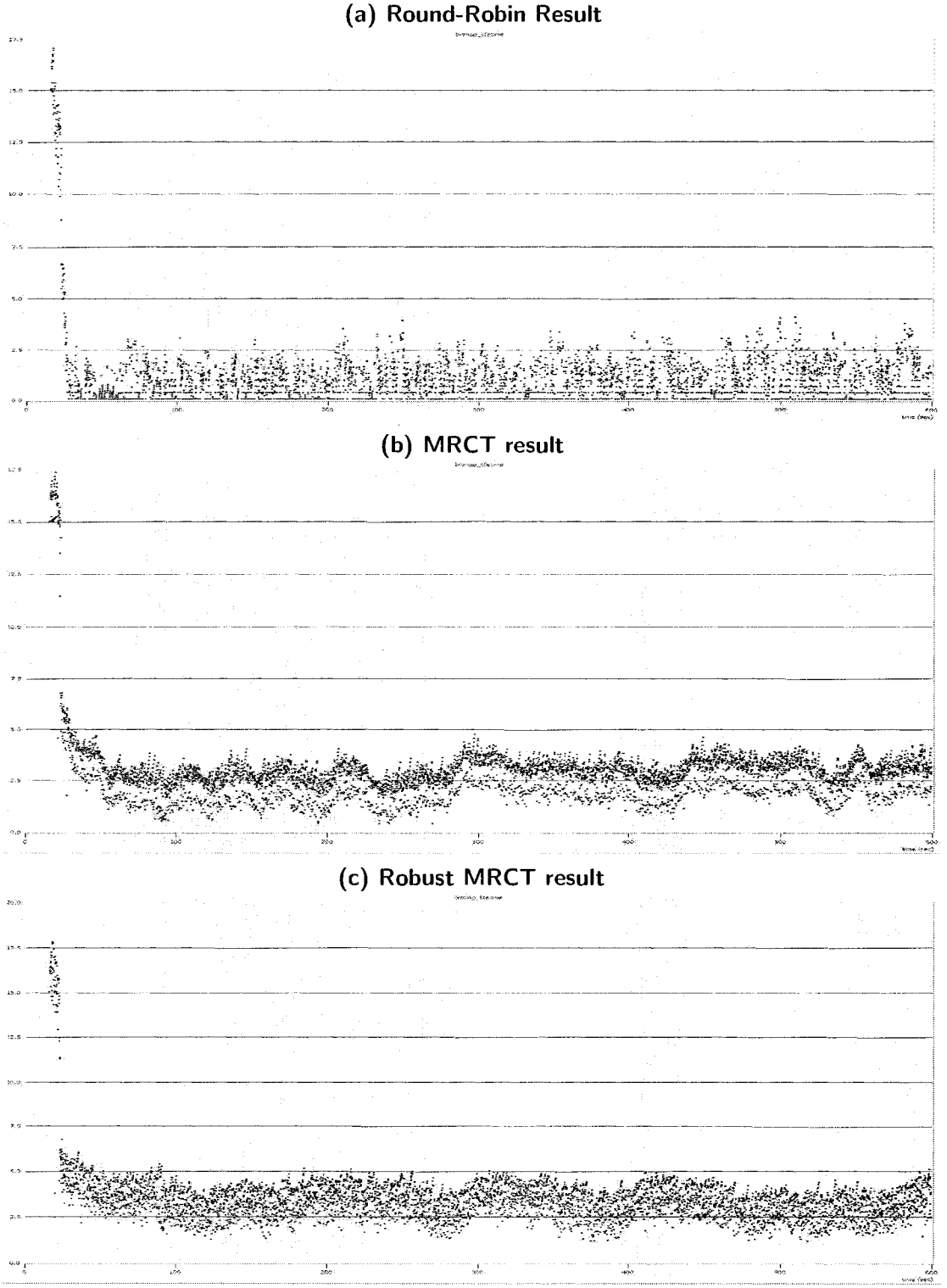


Figure 3.3: Sample plots of the results for the three heuristics (a) round-robin, (b) MRCT, and (c) Robust MRCT.

3.10 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 onto heterogeneous computing systems was studied in [65]. The primary objective in [65] 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 [65] attempting to map each task to its minimum completion time machine. However, the method of computing a completion time in [65] 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 [66], 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 [66], the resource management techniques of our research are appropriate for use in the System Resource Manager role of the mulit-layer resource management architecture.

In [51], 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 environemnt [69]. In [69], 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.

3.11 Summary

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

Stochastic Robustness Metric and its Use for Static Resource Allocations *

4.1 Overview

This research investigates the problem of robust static resource allocation for distributed computing systems operating under imposed Quality of Service (QoS) constraints. Often, such systems are expected to function in a physical environment replete with uncertainty, which causes the amount of processing required over time to fluctuate substantially. Determining a resource allocation that accounts for this uncertainty in a way that can provide a probabilistic guarantee that a given level of QoS is achieved is an important research problem. The stochastic robustness metric proposed in this research is based on a mathematical model where the relationship

*This entire chapter was done jointly with James Smith and appears in the following papers: [86–89].

between uncertainty in system parameters and its impact on system performance are described stochastically.

The utility of the established metric is then exploited in the design of optimization techniques based on greedy and iterative approaches that address the problem of resource allocation in a large class of distributed systems operating on periodically updated data sets. The performance results are presented for a simulated environment that replicates a heterogeneous cluster-based radar data processing center. A mathematical performance lower bound is presented for comparison analysis of the heuristic results. The lower bound is derived based on a relaxation of the Integer Linear Programming formulation for a given resource allocation problem.

4.2 Introduction

Often, parallel and distributed computing systems must operate in an environment replete with uncertainty while providing a required level of QoS. The following are two examples . Fig. 4.1 schematically depicts the Collaborative Adaptive Sensing (CARA) system, which is a joint effort of many technology developers [18]. The CARA example represents a large class of systems that operate on *periodically* updated data sets, e.g., defense surveillance for homeland security, and monitoring vital signs of medical patients. Typically, in such systems, sensors (e.g., radar, sonar, and video camera) produce data sets with a constant period of $\underline{\Lambda}$ time units. Periodic 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 $\underline{N}$ independent applications that can be executed in parallel on the available set of $\underline{M}$ 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.

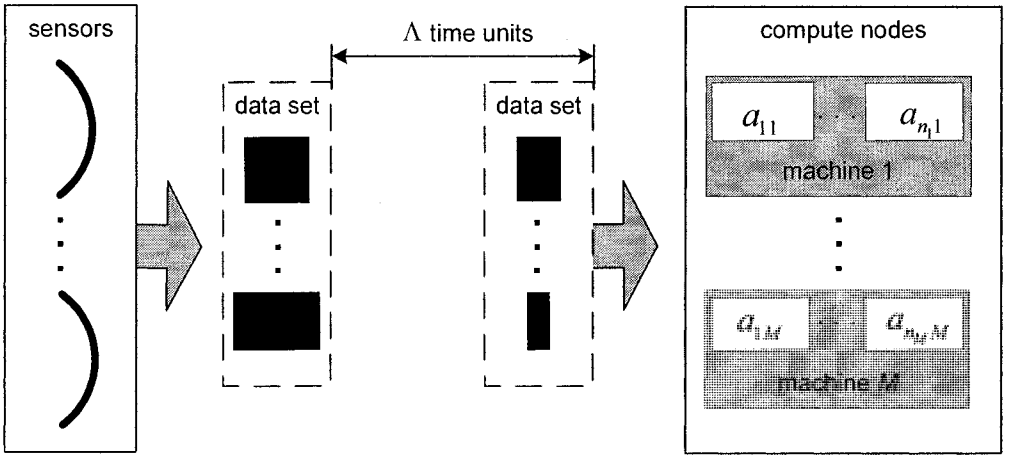


Figure 4.1: The CARA example: major functional units and data flow for a class of systems that operate on periodically updated data sets. The a_{ij}'s denote applications executing on machine j. Processing of each data set must be completed within Λ time units.

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 [11], 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. This list is a result of merging the responses from multiple index servers, each searching over a particular subset of the entire index database. The second phase uses the list of document identifiers and computes the actual title and uniform

resource locator's of these documents, along with any query-specific document summary information. Document servers perform this job, each processing a certain part of the list.

Consider the first phase of the system where a fork-join job [59] must be performed, as shown in Fig. 4.2 (similar analysis can be derived for the second phase). To reduce overall execution time, each query is duplicated and 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 the requests are serviced 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 cluster manager 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.

According to [6], any claim of robustness for a given system must answer three questions: (a) what behavior of the system makes it robust, (b) what uncertainties is the system robust against, (c) quantitatively, exactly how robust is the system? As an example, consider the CARA environment shown in Fig. 4.1, where the system is robust if it is capable of processing each data set within Λ time units. A resource allocation deployed in this system must be robust against uncertainties in execution

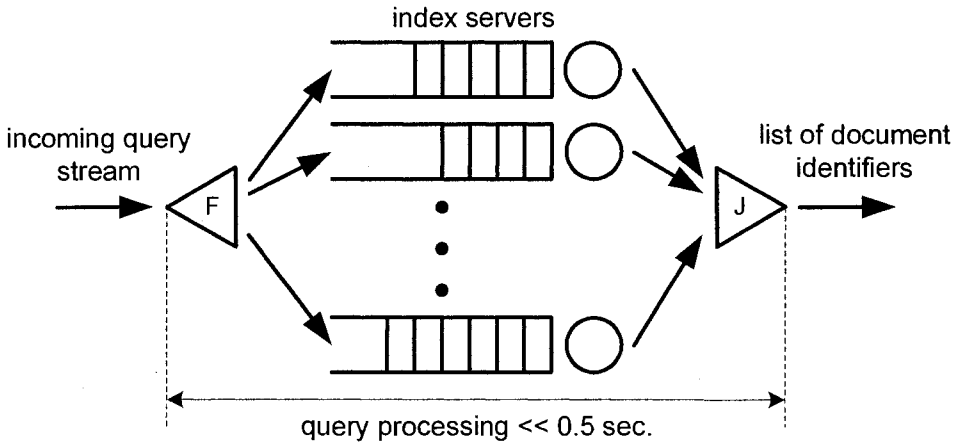


Figure 4.2: The Google example: Fork (F) and Join (J) query processing executed by index servers in the first phase of the search engine.

times of the applications processing data sets. In our approach, the degree of robustness is measured as the probability that all the processing required for a given data set is completed within Λ time units. Very similar definitions could be derived for the Google example.

In both examples, an important task for a resource management system is to distribute applications (or queries) across compute nodes (or index servers) such that the 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 processing times. 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, so tasks can be completed well before their deadlines. However, more sophisticated stochastic analysis is required for resource allocations 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. Our study defines a stochastic methodology for quantifiably determining the ability of a resource allocation to satisfy QoS constraints in the midst of uncertainty in system parameters.

The problem of resource allocation in the field of heterogeneous parallel and distributed computing is NP-complete (e.g., [21, 49]), therefore, the development of heuristic techniques to find near-optimal solutions represents a large body of research (e.g., [4, 16, 32, 34, 35, 49, 60, 65, 99]). There are two major classes of resource allocation approaches widely used in practice: greedy heuristics and iterative algorithms. Usually, greedy heuristics are relatively fast (as opposed to time-consuming global search heuristics), as they generate a solution by making locally optimal decisions; this feature often makes greedy heuristics an appropriate choice to use in dynamic (i.e., on-line mapping) systems. However, the quality of solutions produced by greedy heuristics is generally lower than that produced by global search heuristics that progressively improve a solution through multiple iterations.

In the first part of this chapter, 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 as the probability that the resource allocation will satisfy the expressed QoS constraints. Two alternative means for computing the metric are presented that render the required computation practical in a number of common environments. The utility of the proposed stochastic metric is analyzed in a simulated environment by comparing it against existing deterministic metrics, i.e., metrics where outcomes are not associated with probabilities.

In the second part of this chapter, the proposed method of stochastic robustness

evaluation was integrated into greedy and global search heuristics developed to address the problem of resource allocation for a class of distributed systems operating on periodic data sets schematically depicted in Fig. 4.1. In many systems of the considered class, it is highly desirable to *minimize* the period Λ between subsequent data arrivals while providing a probabilistic guarantee that each data set is processed within Λ time units. As a practical example, consider air traffic control and military applications where frequent radar scans are needed to identify an approaching target with a guaranteed high probability of successful processing of each scan.

In summary, the two major contributions of this chapter include: (1) the development of a mathematical model for a stochastic robustness metric that utilizes available information to quantifiably determine a resource allocation's ability to satisfy expressed QoS constraints; and (2) the design and performance analysis of optimization techniques that solve the problem of robust resource allocation in distributed systems operating on periodically updated data sets. We will show that when the distributions of random variables associated with uncertain parameters in the stochastic model are available, an evaluation of a resource allocation leads to more useful results than that achievable with deterministic metrics utilizing mean values. Additional contributions include a discussion on the applicability of convolution and the bootstrap method for computing the proposed stochastic robustness metric, the derivation of a lower bound on a minimum Λ achievable based on our Integer Linear Programming relaxation, and an analysis of the literature pertinent to the area of robust resource allocation in distributed systems.

In Section 4.3, a formal definition of stochastic robustness is given, while Section 4.4 discusses methods of computing the stochastic robustness metric given the independence of input parameters. A comparison study demonstrating the effectiveness

of the proposed robustness measure versus deterministic metrics is included in Section 4.5. The descriptions of the heuristics or generating a robust resource allocation that utilize the new metric are presented in Section 4.6 and Section 4.7 for greedy and iterative approaches respectively. This is followed by a proof of an effective lower bound in Section 4.8, which is used for comparison in the performance analysis. Section 4.9 contains the details of the simulation setup. The performance results of the developed heuristics are presented in Section 4.10. A discussion of the relation of this study to the published work from the literature is given in Section 4.11. A glossary of notation and acronyms used in the chapter are tabulated in Table 4.1 and Table 4.2, respectively.

Table 4.1: Glossary of Notation

T_{ij}	execution time of application a_{ij} on compute node j
ψ	performance characteristic
ψ_j	local performance characteristic on node j
β_{min}	minimum acceptable value for ψ
β_{max}	maximum acceptable value for ψ
θ	stochastic robustness metric equal to $\mathbb{P}[\beta_{min} \leq \psi \leq \beta_{max}]$
Λ	time period between sequential data sets in the CASA example
N	total number of applications considered for mapping
M	number of heterogeneous nodes in the system

4.3 Mathematical Model for Stochastic Robustness

A stochastic robustness metric for a given distributed computing environment should reasonably predict the performance of the system. Given the existing content in the CASA 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

Table 4.2: Acronyms

CASA	Collaborative Adaptive Sensing of the Atmosphere
FFT	Fast Fourier Transform
PMR	Period Minimization Routine
CR	Contention Resolution heuristic
CRC	Common Stopping Criterion
GA	Genetic Algorithm
ACO	Ant Colony Optimization
SA	Simulated Annealing
LB	Lower Bound
UB	Upper Bound
B&B	Branch-and-Bound algorithm

example, the sequence S_j represents n_j queries assigned to index server j. Let random variable T_{ij} denote the execution time of each individual application (or query) a_{ij} on compute node (or index server) j. The random variables T_{ij} characterize the uncertainty in execution time for each of the applications in the system and serve as the inputs to the mathematical model. These random variables are the uncertainty parameters in the mathematical model.

In the CASA example, the evaluation of system performance is based on the makespan value (total time required for all applications to process a given data set) [16] 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, denoted as ψ, in the model is

$$\psi = \max_{j=1,\ldots,M} \left\{ \sum_{i=1}^{n_1} T_{ij} \right\}. \tag{4.1}$$

In the Google example, the performance in the first phase is measured for each individual query. Unlike the CASA example, where the evaluation of makespan values occurs at each Λ, 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. The functional dependence between the uncertainty parameters and the performance characteristic at time t, denoted as $\psi(t)$, is

$$\psi(t) = \max_{j=1,\ldots,M} \left\{ T_{1j} - (t - t_{0j}) + \sum_{i=2}^{n_j} T_{ij} \right\}. \tag{4.2}$$

Due to its functional dependence on the uncertainty parameters T_{ij}, the performance characteristic in Eq. 4.1 and 4.2 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 system performance [6], i.e., $\beta_{min} \leq \psi \leq \beta_{max}$. **The stochastic robustness metric, denoted as θ, is the probability that the performance characteristic of the system is confined to the interval $[\beta_{min}, \beta_{max}]$, i.e., $\theta = \mathbb{P}[\beta_{min} \leq \psi \leq \beta_{max}]$.** For a given resource allocation, the stochastic robustness quantitatively measures the probability 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.

4.4 Computational Issues

4.4.1 Assumptions of Independence

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 CASA 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 ψ_1, ψ_2, ..., ψ_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

$$\theta = \prod_{j=1}^{M} \mathbb{P}[\beta_{min} \leq \psi_j \leq \beta_{max}]. \tag{4.3}$$

Specifically in Eq. 4.3, $\beta_{max} = \Lambda$ in the CASA example and $\beta_{max} \ll 0.5$ sec. in the Google example. In both examples, β_{min} is set to 0 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, then $\mathbb{P}[\beta_{min} \leq \psi \leq \beta_{max}]$ can be computed using an n_j-fold convolution of probability density functions (pdfs) $f_{T_{ij}}(t_i)$ [62]

$$\mathbb{P}[\beta_{min} \leq \psi_j \leq \beta_{max}] = \int_{\beta_{min}}^{\beta_{max}} [f_{T_{1j}}(t_1) * \ \ldots \ * f_{T_{n_j j}}(t_{n_j})]dt. \tag{4.4}$$

This assumption of independence is valid for non-multitasking execution mode which

is commonly considered in the literature [16,32,59,65,99]), and applied in practice in a variety of systems, e.g., an iterative UDP server model [36].

4.4.2 Fast Fourier Transform Method

An n_j-fold convolution in Eq. 4.4 requires $n_j - 1$ computations of the convolution integral [62]; 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 of T_{ij}, i.e., the forward Fourier transform [76], and $\Phi^{-1}_{\psi_j}$ denotes the inverse Fourier transform, then Eq. 4.4 can be computed as follows

$$\mathbb{P}[\beta_{min} \leq \psi_j \leq \beta_{max}] = \int_{\beta_{min}}^{\beta_{max}} \Phi^{-1}_{\psi_j}\{\Phi_{T_{1j}}(\omega) \times \ \ldots \ \times \Phi_{T_{n_j j}}(\omega)\}dt. \tag{4.5}$$

From this point on, assume that each pdf $f_{T_{ij}}(t_i)$ is expressed as a discrete probability mass function (pmf) utilizing Ω points—this is common in practical implementations. As such, the calculation can be performed using a Fast Fourier Transform method (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 $2\Omega \log \Omega$ [76]. Thus, the computational complexity of determining the local performance characteristic can be drastically reduced, making the approach reasonable to compute.

In dynamic systems (i.e., on-line mapping), 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.

4.4.3 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 [100]. 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, the processing of queries by a Web server is typically done in a parallel multitasking environment, and there exists a complex functional dependence [8] between the time required to process a query and a number of currently executing threads, amount of data cached, types of requests, etc.

Suppose that for each T_{ij}, its execution time distribution is known and fully described with a pmf $f_{T_{ij}}(t_i)$. The pmf can be derived analytically and presented as a closed-form expression, or obtained as a result of past executions of application i on compute node j. The latter is called a sample pmf. As a number of past executions k grows, new results of executions 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 $\widehat{T}^*_{ij}$ denote one draw from the distribution $f_{T_{ij}}(t_i)$ (or from $\widehat{f}_{(k)T_{ij}}(t_i)$). Let $\widehat{\psi}^*_j$ be a bootstrap replication whose computation is based on a known functional

dependence $\underline{g()}$ between T_{ij} and ψ_j, i.e., $\widehat{\psi}_j^* = g(\widehat{T}_{1j}^*, ..., \widehat{T}_{n_j j}^*)$. In the bootstrap simulation step [100], $\underline{B}$ bootstrap replications of $\widehat{\psi}_j^*$ can be computed, $\widehat{\psi}_{j,1}^*, ..., \widehat{\psi}_{j,B}^*$, and used to approximate a pmf of ψ_j, denoted as $\underline{\widehat{f}_{(B)\psi_j}(t)}$. Thus, the probability for the local performance characteristic ψ_j can be approximated as:

$$\mathbb{P}[\beta_{min} \leq \psi_j \leq \beta_{max}] \approx \int_{\beta_{min}}^{\beta_{max}} \widehat{f}_{(B)\psi_j}(t)dt. \tag{4.6}$$

Eq. 4.6 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 [100]. An exact normalizing transformation will rarely exist, but approximate normalizing transformations may exist —which causes the probability that ψ_j is in the interval $[\beta_{min}, \beta_{max}]$ to be not exactly equal to the integral on the right-hand side of Eq. 4.6. The pseudocode for the bootstrap analysis is shown in Fig. 4.3.

```
B ← number of bootstrap replications;
Vboot ← vector of length B;
Vsample ← vector of length n_j;
for b = 1 : B
   for i = 1 : n_j
      Vsample ← sample f_{T_ij}(t_i) with replacement;
   Vboot ← g(Vsample);
   clear Vsample;
Nsamples ← number of samples in Vboot ∈ [β_min, β_max];
P[β_min ≤ ψ_j ≤ β_max] ≈ Nsamples/B.
```

Figure 4.3: Pseudocode for the bootstrap procedure.

Table 4.3 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 is a summation. Table 4.3 captures the percent error resulted from the approximations based on Eq. 4.6 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 in $\widehat{f}_{(B)\psi_j(t)}(t)$—this ensures that β_{max} is specified in the reasonable range. All T_{ij} distributions were modeled by randomly assigning a probability to each of Ω data points and normalizing the resultant pmfs. Each value in Table 4.3 represents the average across 100 different trials. Two trends can be identified from Table 4.3: (1) relative accuracy does not increase with 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 Gaussian distribution —which occurs often in practice— the resultant bootstrap approximations would be more precise as described in the proof of Eq. 4.6 [100]. There are other bootstrap approximations that may be more accurate, especially when the nature of the expected pmf of the performance metric is known. The above experiment demonstrates that the bootstrap method is capable of producing reasonable approximations. The real strength of the bootstrap is its capability of handling mutually dependent random variables. Note however that some bootstrap methods require a significant amount of computation and might be prohibitively expensive in certain distributed systems.

Table 4.3: Percent error resulted from 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

4.5 Comparison with Deterministic Metrics

The experiments in this section seek to establish the utility of the stochastic robustness metric in distinguishing between resource allocations that perform similarly with respect to a commonly used deterministic metric, such as makespan, and the deterministic robustness metric from [6]. The simulation of the system outlined in the CASA example of Section 4.2 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 pmf randomly constructed on the range $[0, 40]$ seconds, inclusive. To construct each pmf, ten execution time values were uniformly spread across the range of the distribution. Each of these execution time values was assigned a probability sampled uniformly on the range $(0, 1)$. All application execution time distributions were subsequently normalized so the sum of the probabilities across all the execution time values becomes equal to 1. Let $\underline{mean_{av}}$ be the average value computed across the means of all constructed application execution time distributions. In the simulation, the QoS constraint Λ was set to $\Lambda = 1.5 \times N \times mean_{av}/M$. Recall, for the CASA example Λ is a QoS constraint on system processing time that is used in the definition of the stochastic robustness metric given in Eq. 4.3. In Fig. 4.4, 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. 4.4(a), a comparison between the stochastic robustness metric and makespan is presented for 1000 *randomly* generated resource allocations. As can be expected, in general, resource allocations that produce a very large makespan tend to have a very

small stochastic robustness metric value. However, there can be a large discrepancy between the predicted performance found using the predicted makespan, based on execution time mean values, and the predicted performance found 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 predicted 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, whereas 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.

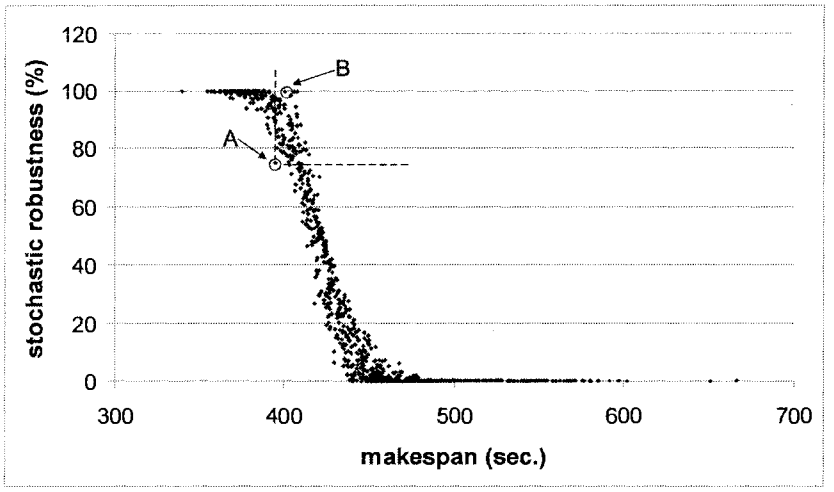
In Fig. 4.4(b), a comparison of the stochastic robustness metric and the deterministic robustness metric is presented for 1000 randomly generated resource allocations. The deterministic robustness metric, first introduced in [6], is based on a calculation of the minimum total increase across all task execution times in the Euclidean sense that can possibly violate Λ. 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.

Compare the two resource allocations C and D. Based on using deterministic robustness measure, 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

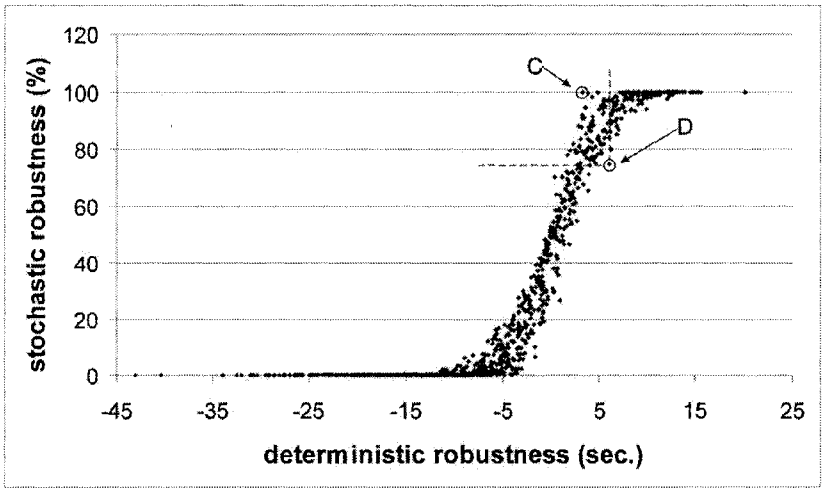
D (with a stochastic measure of 75%). Thus in this case, using only the deterministic robustness metric to select a resource allocation, D appears to be more robust than C. In contrast, the stochastic robustness metric, which accounts for the distribution of makespan outcomes, shows that allocation C has a 99.9% probability of meeting the QoS constraint while allocation D has only a 75% probability of meeting the QoS constraint.

Consider the sub-region identified in Fig. 4.4(b) with dotted lines originating from the point D, containing all of the points above and to the left of D. Each of these points in the sub-region has a higher stochastic robustness metric value than D but a lower deterministic robustness metric value than D.

It is shown in [6] that the deterministic robustness metric, using an expected time for each task execution, provides better information for resolving a resource allocation than just a makespan. However, when execution time distributions are available, the stochastic robustness metric provides even better decision than the deterministic robustness metric. 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. 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.



(a)



(b)

Figure 4.4: 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 difference between the stochastic robustness metric and makespan. Similarly, the stochastic robustness metric values for allocations C and D exemplify the difference with the deterministic robustness metric.

4.6 Greedy Heuristics

4.6.1 Overview

This research assumes that an acceptable level of stochastic robustness $\mathbb{P}[\psi \leq \Lambda]$ is specified for the system described in the CASA example in Section 4.2. Thus, the performance goal for the mapper is to find resource allocations for a given set of N applications on M machines that allows for the minimum period Λ between sequential data sets while maintaining a given level of stochastic robustness.

Four greedy heuristics were designed for the problem of finding a resource allocation with respect to this objective. Greedy techniques have been adapted in many systems, e.g., [16, 49, 70, 73], as they perform well and are capable of generating solutions relatively fast as compared to time-consuming global search heuristics, e.g., [99, 101, 102]. The four heuristics can be categorized based on the amount of stochastic information that each of them uses. The first two of the proposed heuristics utilize the entire spectrum of stochastic information at each stage of the decision process, as opposed to the third heuristic that uses mean values in the sorting stage, and the fourth heuristic that operates using mean values only. All of the heuristics employ the Period Minimization Routine, described next, to determine the minimum Λ supported by each resource allocation.

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 the n_j-fold convolutions are obtained with the FFT or bootstrap method 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 on compute node j is comprised of K_j impulses, where every impulse is specified by

$lo = t_1 \leftarrow \min\{t_{kj} \mid 1 \leq k \leq K_j, 1 \leq j \leq M\}$;
$hi = t_2 \leftarrow \max\{t_{kj} \mid 1 \leq k \leq K_j, 1 \leq j \leq M\}$;
$P \leftarrow$ 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 4.5: Pseudocode for the Period Minimization Routine (PMR).

the time $\underline{t_{kj}} \mid k \in [1, K_j]$, and the probability $\underline{p_{kj}} \mid k \in [1, K_j]$ for t_{kj} to occur.

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 $\prod_{j=1}^{M} \left[\sum_{k=1}^{K_j} p_{kj} \times \mathbf{1}(t_{kj} \leq \Lambda) \right]$, where $\mathbf{1}(condition)$ is 1 if *condition* is true; 0 otherwise. The PMR procedure is summarized in Fig. 4.5.

After $\underline{Q}$ steps, the PMR procedure reduces the uncertainty range by a factor $\approx (0.5)^Q$, which is the fastest possible uncertainty reduction rate. This optimality is possible because $\mathbb{P}[\psi \leq \Lambda]$ is strictly increasing as the number of impulses considered for its computation grows. The notation $\underline{\Lambda(a_i, m_j)}$ will be used to denote a PMR call that returns the minimum value of Λ for the specified level of stochastic robustness when application a_i is added to machine m_j.

while not all applications are mapped
  for each unmapped application a_i find the compute node m_j such that
    $m_j \leftarrow$ argmin* $\{\Lambda(a_i, m_j) \mid 1 \leq j \leq M\}$;
    resolve ties arbitrarily;
  from all (a_i, m_j) pairs found above select pair(s) (a_x, m_y) such that
  $(a_x, m_y) \leftarrow$ argmin $\{\Lambda(a_i, m_j) \mid$ all $(a_i, m_j)\}$;
  resolve ties arbitrarily;
  map a_x on m_y;
end of while

Figure 4.6: Pseudocode for the two-phase Basic greedy heuristic.

4.6.2 Basic Heuristic

The Basic heuristic is based on the principles of the Min-Min algorithm (first presented in [49], and shown to perform well in many environments, e.g., [16, 65, 70]). The heuristic traverses through N iterations resolving an allocation of one application at each iteration. In the first phase of each iteration, the heuristic determines the best assignment (according to the performance goal) for each of the applications left unmapped. In the second phase, it selects which application to map based on the best result found in the first phase. The Basic procedure is summarized in Fig. 4.6.

4.6.3 Contention Resolution Heuristic

The CR heuristic uses the sufferage concept introduced in [65], and used in [57]. Like the Basic heuristic, in every iteration this heuristic first determines the best assignment for each of the applications left unmapped. Mapping decisions are finalized

*Argmin stands for the argument of the minimum, i.e., the value of the given argument for which the value of the given expression attains its minimum value

for those applications whose best choice compute nodes are unique, i.e., there are no other applications competing for these nodes. In the second phase, the most critical among the competing applications gets allocated, determined as the application with the largest difference between the two smallest Λ values corresponding to this application's assignment to its best choice and its second best choice compute nodes, i.e., its sufferage. The CR procedure is summarized in Fig. 4.7.

while not all applications are mapped
  for each unmapped application a_i
    find the first choice compute node $m_j^{(1)}$ as
    $m_j^{(1)} \leftarrow \text{argmin}\{\Lambda(a_i, m_j) \mid 1 \leq j \leq M\}$;
  for all $(a_i, m_j^{(1)})$ pairs found above
    if $m_j^{(1)} \neq m_k^{(1)}$, $\forall k \neq j$, **then** map a_i on $m_j^{(1)}$;
    else find the second choice compute node $m_j^{(2)}$ as
      $m_j^{(2)} \leftarrow \text{argmin}\{\Lambda(a_i, m_j) \mid (1 \leq j \leq M \& \; m_j \neq m_j^{(1)})\}$;
      compute contention value $C(a_i)$ as
      $C(a_i) \leftarrow \{\Lambda(a_i, m_j^{(2)}) - \Lambda(a_i, m_j^{(1)})\}$;
  select unmapped application a_x with maximum $C(a_i)$ as
  $a_x \leftarrow \text{argmax}\{C(a_i) \mid \text{all unmapped } a_i\}$;
  resolve ties arbitrarily;
  map a_x on its first choice compute node $m_j^{(1)}$;
end of while

Figure 4.7: Pseudocode for the two-phase Contention Resolution greedy heuristic.

4.6.4 Sorting Heuristic

This heuristic uses the concepts developed for the MCT algorithm that were observed to perform well in multiple resource allocation schemes designed for distributed systems, e.g., [16, 65]. Initially, all N applications considered for mapping are sorted

based on the average computed for each application across the mean values $\mu(T_{ij})$ derived from execution time distributions of $T_{ij} \mid 1 \leq j \leq M$. Three different orderings were considered in the experiments. The HI→LO ordering where applications are ranked in descending order of their averages; LO→HI ordering where applications are ranked in ascending order of their averages; and ARBITRARY ordering where N applications are ordered randomly. Once sorting is completed, applications are fetched sequentially, each mapped on the compute node selected to provide the minimum value of the period Λ under the imposed level of stochastic robustness. The heuristic's procedure is summarized in Fig. 4.8.

for each application a_i

find the average $A(a_i) \leftarrow \left[\sum_{j=1}^{M} \mu(T_{ij})\right] / M$;

order applications based on their averages $A(a_i)$

according to the selected type {HI→LO, LO→HI, ARBITRARY};

while not all applications are mapped

fetch next unmapped application a_i from the ordered list;

find the compute node m_j such that $m_j \leftarrow \text{argmin}\{\Lambda(a_i, m_j) \mid 1 \leq j \leq M\}$;

resolve ties arbitrarily;

map a_i on m_j;

end of while

Figure 4.8: Pseudocode for the Sorting greedy heuristic.

4.6.5 Mean Load Balancing Heuristic

This heuristic was developed based on the concepts of the OLB algorithm discussed in [56, 65]. First, the N applications are sorted based on average value, as in the

sorting heuristic. Then, the applications are mapped in the {HI→LO, LO→HI, ARBITRARY} order where the compute node with the minimum mean of its execution time distribution is selected for each allocation. The heuristic's procedure is summarized in Fig. 4.9.

for each application a_i
    find the average $A(a_i) \leftarrow \left[\sum_{j=1}^{M} \mu(T_{ij}) \right] / M$;
order applications based on their averages $A(a_i)$
according to the selected type {HI→LO, LO→HI, ARBITRARY};
while not all applications are mapped
    fetch next unmapped application a_i from the ordered list;
    find the compute node m_j such that $m_j \leftarrow \text{argmin}\{\mu(T_{ij}) \mid 1 \leq j \leq M\}$;
    resolve ties arbitrarily;
    map a_i on m_j;
end of while

Figure 4.9: Pseudocode for the Mean Load Balancing greedy heuristic.

4.7 Global Search Heuristics

4.7.1 Overview

Three global search heuristics were designed to find a resource allocation that optimizes the performance goal stated in Subsection 4.6.1. These heuristics are probabilistic search techniques that have been widely used in optimization research [70,85,102], artificial intelligence [47], and many other areas. The first two of the heuristics operate with a set of complete resource allocations; whereas the third heuristic iteratively changes a single complete resource allocation. As opposed to the previous greedy

algorithms, where a single complete resource allocation was "constructed," iterative heuristics progress toward a final solution through modified versions of complete resource allocations. During each iteration, the existing complete resource allocation (or a set of 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 global search heuristics and to demonstrate the performance over time for each of them, a common stopping criterion (CSC) of 150,000 calls to the PMR routine was used in this study. It is important to note that the PMR stochastic evaluation is the most computationally intensive part of any of the algorithms as it calls for M executions of $(n_j - 1)$-fold convolutions, followed by a recursive search for a minimum Λ level.

4.7.2 Steady State Genetic Algorithm

The adapted genetic algorithm (GA) implementation was motivated by the Genitor evolutionary heuristic [102]. 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. The 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 members 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 greedy Sorting heuristic presented before, in which the ARBITRARY ordering among applications was perturbed to have as a result different resource allocations to serve as the initial members of the population. In addition, the solution

produced by the greedy Basic heuristic was also added to the initial population.

The GA was implemented as a *steady state* GA, i.e., for each iteration of the GA only a single pair of chromosomes was selected for crossover. Selection for crossover was implemented as rank-based selection using a linear bias function [102] where the population of chromosomes is sorted by Λ values. 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 its *Lambda* value such that after insertion the population remains sorted. Furthermore, the population is truncated after insertion to maintain a constant 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 or the graveyard and the time required to evaluate the new chromosome. The graveyard size was limited to 20,000 chromosomes.

To maintain the selective pressure of rank-based selection, an additional constraint was placed on the population requiring each chromosome to be unique, i.e., clones are explicitly disallowed. If a chromosome produced in any iteration 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 for insertion into the population.

The crossover operator was implemented using a two-point reduced surrogate procedure [102] where the elements between the crossover points are exchanged between the two parents. Crossover points are selected such that at least one element of the parent chromosomes differs between the selected crossover points as this guarantees

offspring that are not clones of their parents. In addition, each generated offspring is checked for uniqueness in the population and graveyard prior to making a call to the PMR routine that calculates the minimum Λ value.

The final step in a single iteration of the GA is mutation. For each iteration of the GA, the mutation operator is applied to the newly generated offspring of the crossover operator. Each application assignment of the offspring is individually mutated with a 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 chosen application, the mutation operator randomly selects a different compute node assignment from a subset of compute nodes that provide smallest means of execution times. The best results in the simulation study were achieved when the size of this subset was set to three. Following mutation a final local search procedure, conceptually analogous to the steepest descent technique, was applied to the result prior to inserting the mutated chromosome into the population.

The local search operator was introduced for inclusion into a GA as a follow-on step to the mutation operator for a flowshop problem in [107]. The implementation of the local search procedure, is similar to the coarse refinement presented as part of the GIM heuristic described in [97]. In particular, all applications are examined to determine which of them should be moved to a different compute node to realize the largest decrease in the minimum supportable Λ value. The procedure continues until moving any application would result in an increase in the minimum supportable Λ value.

The GA procedure is summarized in Fig. 4.10.

```
generate initial population;
evaluate each chromosome;
rank population based on Λ values;
while CSC not met
   select two chromosomes from the population;
   select crossover points;
   exchange compute node assignments
   between crossover points;
   ascertain if either offspring are unique;
   for each element of each child chromosome
     generate a random number x in the range [0,1];
     if x < mutation rate
        determine 3 minimum mean execution time machines
        for the selected application;
        arbitrarily change the compute node assignment
        of the selected application;
   apply local search to each of the offspring;
   ascertain if either offspring 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 4.10: Pseudocode for the Steady State Genetic Algorithm.

4.7.3 Ant Colony Optimization

The Ant Colony Optimization (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 [33]. 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 each element of the pheromone table, denoted as $\tau(a_i, j)$, represent the "goodness" of compute node 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 common 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) the performance metric and (b) the pheromone table information. The ant procedure involves two phases. In Phase 1, for each unmapped application, the compute node, denoted as $j_{best}(a_i)$, is determined such that it would provide the minimum mean completion time, $\mu_{min}(a_i)$,

across all M compute node completion time distributions. Each of these distributions is obtained by mapping a_i to the compute node and determining the new completion time distribution for the compute node. The worth of application a_i, denoted as $\eta(a_i)$, is then determined as a result of the following normalization

$$\eta(a_i) = \frac{\mu_{min}(a_i)}{\sum_{\text{unmapped } a_k} \mu_{min}(a_k)}. \tag{4.7}$$

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

Let the fitness of ant s, denoted as $f(s) \in (0, 1)$, be determined as the 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 with a PMR call invoked at the end of each ant procedure, and ranking is done using a linear bias function [102]. 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 $\tau(a_i, j)$ is updated as follows

$$\tau(a_i, j) = \rho \times \tau(a_i, j) \ + \sum_{s=1}^{Q} f(s) \times \mathbf{1}(a_i \text{ assigned to } j \text{ in } B_s). \tag{4.8}$$

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

$$\mathbb{P}[a_i \text{ selected next}] = \frac{\alpha \times \tau(a_i, j_{best}(a_i)) + (1-\alpha) \times \eta(a_i)}{\sum_{\text{unmapped } a_k} \alpha \times \tau(a_k, j_{best}(a_k)) + (1-\alpha) \times \eta(a_k)}. \tag{4.9}$$

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. Note that numerical values for all of the aforementioned parameters were determined with respect to the input specified for the conducted experiments—i.e., these values must be readjusted for different inputs.

The ACO procedure is summarized in Fig. 4.11.

```
initialize pheromone table;
while CSC not met
    for each ant
        while there are unmapped applications
            select application a_i according to Eq. 4.9;
            map application a_i to j_best(a_i) compute node;
            break ties arbitrarily;
        end of while;
        compute f(s) via PMR call;
    update pheromone table according to Eq. 4.8;
end of while
map each application a_i to its j_best(a_i) compute node.
```

Figure 4.11: Pseudocode for the Ant Colony Optimization.

4.7.4 Simulated Annealing

The Simulated Annealing (SA) algorithm —also known in the literature as Monte Carlo annealing or probabilistic hill-climbing [70]— is based on an analogy taken from thermodynamics. In SA, a randomly generated solution, structured as the chromosome for GA, is 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 applies the mutation operation in the same fashion as GA including the local search. Once a new *unique* solution, denoted as S_{new}, is produced (SA uses the same graveyard technique as GA to determine uniqueness), a decision regarding the replacement of a previous solution with a new one has to be made. If the quality of the 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 [70]. This probability 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, in the latter case, the SA algorithm selects a sample from the range $[0, 1)$ according to a uniform distribution. If

$$random[0,1) > \frac{1}{1+\exp^{(\frac{\Lambda(S_{old})-\Lambda(S_{new})}{\mathbb{T}})}} \tag{4.10}$$

the new poorer resource allocation is accepted; otherwise, the old one is kept. As it follows from Eq. 4.10, 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 99% of its current value. This percentage, defined as a cooling rate, was determined experimentally by varying the rate in the range of $(0.9, 1]$ in 0.01 steps. The initial system temperature in Eq. 4.10 was set to Λ of the chosen initial resource allocation.

The SA procedure is summarized in Fig. 4.12.

```
S_old ← initial randomly generated resource allocation;
𝕋 ← Λ(S_old);
while CSC not met
    S_new ←result of successful mutation;
    if Λ(S_new) < Λ(S_old)
        S_old ← S_new;
    else if Eq. 4.10 holds
            S_old ← S_new;
𝕋 ← 0.9 × 𝕋;
end of while
```

Figure 4.12: Pseudocode for the Simulated Annealing.

4.8 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., $\theta \geq 0.5$, which is typical for practical implementations. The

process of calculating the 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 resource allocation problem.

Step 1: Consider a *given* complete resource allocation of N applications on M compute nodes. Let $\underline{\overline{\Lambda}}$ denote the maximum of the means across all M completion time distributions, $\underline{\mu(\sum_{i=1}^{n_j} T_{ij})}$, 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 Eq. 4.2, $\mathbb{P}[\psi \leq \overline{\Lambda}]$ 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 $\Lambda^* \leq \min\{\overline{\Lambda} \mid \text{ all possible mappings}\}$. Relying on the property that the sum of means is equal to the mean of the sums, i.e., $\sum_{i=1}^{n_j} \mu(T_{ij}) = \mu(\sum_{i=1}^{n_j} T_{ij})$, 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_i is assigned to compute node η_j, and equal to zero if a_i is not assigned to compute node η_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 ILP formulation presented below can easily be converted to a canonical ILP form [20].

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 the Branch-and-Bound technique). However, condition (b) makes the ILP form NP-complete [71], so that for large-scale problems a Linear Programming (LP) relaxation is required to the ILP form that implies that condition (a) is relaxed to real numbers, i.e., $x[i,j] \in [0,1] \mid \{1 \leq i \leq N, 1 \leq j \leq M\}$. Due to this relaxation, in general, an LP solution does not correspond to a valid mapping, but allows a global optimal solution to be found in *polynomial* time [44], that will be a lower bound for the ILP global optimal solution Λ^*. Note that the derived LB is tighter for stochastic robustness levels approaching 0.5; this is a result of using mean values in the LB computation.

4.9 Simulation Setup

To evaluate the performance of the heuristics described above for the considered class of distributed HC systems operating on periodic data, the following approach was used to simulate a cluster-based radar system schematically illustrated in Fig. 4.1. 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 with benchmark results. The experimental data, represented by two execution time sample pmfs, were obtained by conducting experiments on the Colorado MA1 radar [52]. These sample pmfs contain times taken to process

500 radar rays of different complexity by the Pulse-Pair & Attenuation Correction algorithm [13] and by the Random Phase & Attenuation Correction algorithm [13], 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[‡] based on the results of the fourteen floating point benchmarks from the CFP2000 suite [95]. Combining the results available from the 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 ij^{th} element corresponds to the execution time distribution of a possible ray processing algorithm of type i on compute node 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 heuristic.

4.10 Experimental Results

4.10.1 Greedy Heuristics

The results of our experiments with the Greedy heuristics are presented in Fig. 4.13. Both two-phase heuristics perform comparably and significantly outperform the one-phase heuristics. By utilizing the entire spectrum of stochastic information at each stage of the decision process these two heuristics are able to outperform the others,

[‡]The eight compute nodes selected to be modeled were: Altos R510, Dell PowerEdge 7150, Dell PowerEdge 2800, Fujitsu PRIMEPOWER650, HP Workstation i2000, HP ProLiant ML370 G4, Sun Fire V65x, and Sun Fire X4100.

in terms of minimizing Λ.

All of the variants of the one-phase sorting heuristic (the results for ARBITRARY ordering represent the average obtained over 50 reshuffled application orderings) performed consistently better than the mean load balancing heuristic variants but worse than two-phase heuristics. Recall that the sorting algorithm utilizes all of the available stochastic information to select individual task machine pairings but relies on deterministic information to order tasks for their selection. By utilizing a task ordering process that relies on deterministic information only, the number of required convolutions to produce a mapping is drastically reduced but the quality of the mapping is also affected. For example, the first two-phase heuristic required approximately 66,000 1-fold convolutions to produce a mapping, whereas the one-phase sorting heuristic required only 1024 1-fold convolutions to construct a mapping. This difference in the number of convolutions directly translated into a roughly 30 times reduction in the execution time of a simulation trial using the latter heuristic.

Finally, the one-phase mean load balancing heuristic consistently performed the worst because it ignores the available stochastic information about task execution times. This results in ignoring the impact of machine heterogeneity on the completion time distributions, which is reflected in a high Λ value. Because the one-phase mean load balancing heuristic only operates with the means of execution time distributions during the mapping process, this heuristic avoided time-consuming convolution calls. This enabled Mean Load Balancing heuristic to finish in a small fraction of the time required for either two-phase heuristic to generate a mapping.

Once the simulation results had been collected for the developed heuristics, it was noticed that there was a large discrepancy in the amount of computation required to produce each of the various mappings, i.e., two-phase heuristics required

tens of thousands of convolutions to produce a mapping as opposed to one-phase techniques required 1024 or less. Consequently, two new variants of the one-phase greedy algorithms that use multiple iterations, denoted in Fig. 4.13 as ITERATIVE, were created to increase the number of evaluated solutions to the level of Basic and CR, i.e., enable these variants to utilize roughly the same amount of computation to produce a mapping.

In both iterative greedy variants, a random restart step was introduced so that after a mapping is produced a new random ordering is generated and the heuristic is executed again. Upon completion of each iteration the resultant mapping is compared against the best mapping found so far by previous iterations. If the new mapping is an improvement on the best mapping, then it is retained as the new best mapping, otherwise it is discarded.

The results of the iterative variants are plotted in Fig. 4.14. As can be expected, the results of both iterative greedy approaches demonstrated some improvement over their non-iterative versions. However, the iterative version of the Sorting greedy heuristic performed worse than the Basic heuristic (the confidence intervals of the two do not overlap) but is a marked improvement over the corresponding non-iterative greedy version. The average Λ over 50 trials of the Basic heuristic was 542.5 msec.; whereas the average Λ over 50 trials of the iterative version of the Sorting greedy heuristic was 569.7 msec.—each had a confidence interval of 7 msec. The performance demonstrated by the iterative version of the Mean Load Balancing heuristic was still significantly worse than the performance of the other heuristics.

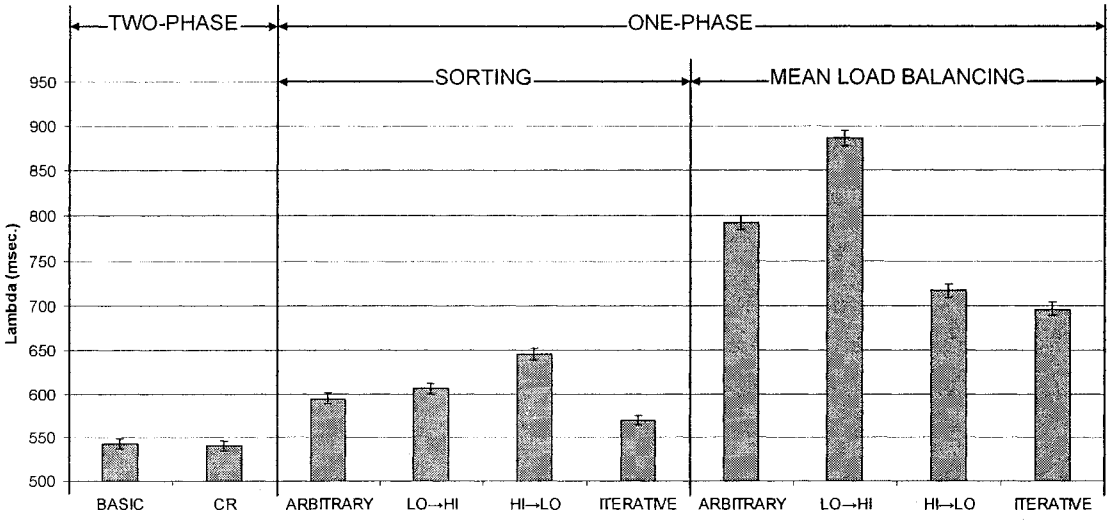


Figure 4.13: A comparison of the results obtained for the described heuristics where the minimum acceptable robustness value was set to be 0.90. The y-axis corresponds to a Λ value obtained by executing the corresponding heuristics. The Λ value for each heuristic corresponds to the average over 50 trials.

4.10.2 Global Search Heuristics

The results of the simulation are presented in Fig. 4.14. Both the GA and SA heuristics were able to improve upon the results of the Basic heuristic of [88] 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 results such that the confidence intervals of the ACO and Basic results are overlapping.

Across the 50 trials tested, LB produced a mean minimum supportable Λ of 469.8 msec. The mean of the Basic heuristic over the same 50 trials was found to be 542.5 with a 95% confidence interval of 7.07. The ACO results 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

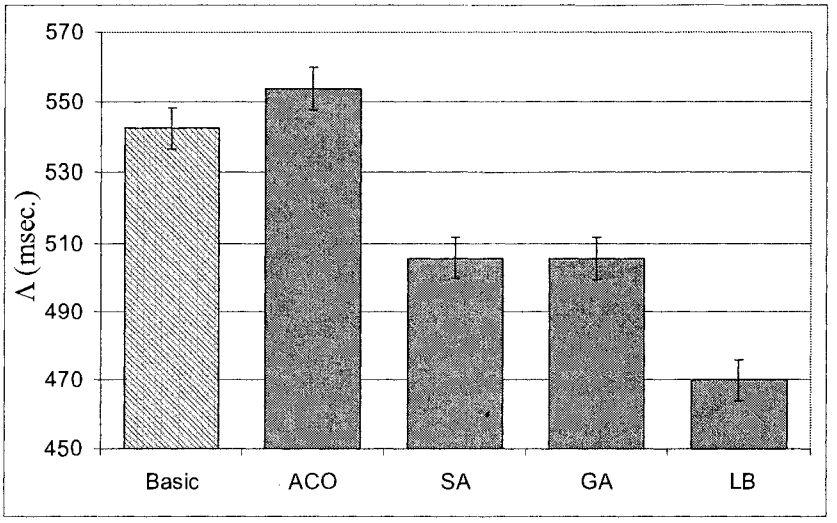


Figure 4.14: A comparison of the results obtained for the described heuristics where the minimum acceptable robustness value was set to be 0.90. The y-axis corresponds to a Λ value obtained by executing the corresponding heuristics. The Λ value for each heuristic corresponds to the average over 50 trials, while the error bars correspond to 95% confidence intervals.

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 ACO heuristic was unable to improve upon the results of the Basic heuristic. 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 [88], 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 other distributed systems. 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.

4.11 Related Literature

In heterogeneous distributed systems the concept of robust resource allocation called for a foundation of a universal robustness framework. The latter issue was first addressed in [6] -- prior work 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 [6], 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 [6], 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 [6], 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 [6] 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 the FFT and bootstrap approximation methods that greatly simplify the required calculations.

In [15], 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 [12,27,64]. In [12], 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 [50], 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 [6] was used in multiple heuristics approaches presented in [97]. 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 evolutionary algorithms, e.g., Genitor and Memetic Algorithm, demonstrate higher performance as compared to the non-iterative greedy heuristics. However, 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 our study, greedy heuristics applied in a stochastic domain experienced a significant execution slowdown due to a substantial number of calls for a convolution routine required at each step of a mapping "construction" process. Although this indicate that greedy heuristics may be inappropriate choices for systems where the time allotted to produce a mapping is strictly limited, the application of a bootstrap approximation method presented in Section 4.3 can alleviate such a problem in some of those systems.

In [32], 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 in the stochastic domain and used to minimize the expected system makespan given a stochastic model of task execution times, i.e., the fitness metric there was based on the first moment of random variables. As shown, this approach works well for unconstrained optimization problems; however, in our study, the imposed QoS constraint in the distributed system

makes the optimization problem constrained calling for other methods. Therefore, our 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, to compute which the entire spectrum of stochastic information needs to be utilized.

4.12 Summary

This chapter proposes a stochastic framework that allows for evaluation and generation robust resource allocations in distributed heterogeneous computer systems operating in uncertain environments. As a basis for this framework, a new stochastic robustness metric was established mathematically. Given the raw volume of computation required to compute this metric, the bootstrap approximation and FFT computational methods were explored to aid the practitioner to apply this approach in different real world scenarios. A utility of the new metric was evaluated in the simulated environment based on distinguishing among resource allocations that perform similarly with respect to a commonly used deterministic metric, such as a makespan, and the deterministic robustness metric presented in [6].

In the second part of this chapter, the new stochastic robustness metric was integrated into a set of greedy and global search heuristics designed for a large class of heterogeneous clusters operating on periodic data sets. The goal in the experiments was to generate a resource allocation that allows for the minimum time period between sequential sensor outputs in a simulated radar system and guarantees a specified level probability that data processing is completed in time.

The Basic and CR two-phase greedy heuristics developed in this study utilized

the entire spectrum of the available stochastic information. These heuristics significantly outperformed Sorting and Mean Load Balancing heuristics, as the stochastic information in the last two was replaced with mean values completely or in the first phase. Furthermore, greedy heuristics were rather time-consuming when applied in the stochastic domain due to multiple calculations of the resultant probability mass functions. Thus, it was reasonable to compare the performance of the greedy heuristics against global search algorithms. Three global search algorithms adapted in this study, i.e., GA, SA, and ACO, were tested under the same stopping criterion. Multiple parameters pertaining to each algorithm were setup for the highest efficiency in a given environment. A comparison analysis against the best greedy results and the lower bound, obtained by solving the relaxed ILP form, revealed a great potential of the GA and SA algorithms to manage efficiently resource allocations in distributed heterogeneous systems operating under uncertainty.

Chapter 5

Sequential Resource Allocation in Distributed Systems under Random Node Failures

5.1 Overview

The problem of finding efficient workload distribution techniques is becoming increasingly important today due to the proliferation of parallel architectures in heterogeneous distributed systems. In many practical cases, the availability of compute nodes in a system changes spontaneously over time, i.e., nodes may leave or fail, join or recover in a random fashion. Therefore, the resource-allocation policy must be designed to be robust with respect to absence and re-emergence of compute nodes so that the performance of the system is maximized over a wide range of processor and task heterogeneity. Such a policy is developed in this work, and its performance is evaluated on a model of a dedicated system composed of a limited set of heterogeneous Web

servers. Assuming that each HTML request results in a "reward" if completed before its hard deadline, the goal in this policy is to maximize a cumulative reward for successfully processed HTML requests. A failure rate for each server is set relatively high to simulate its operation under harsh conditions. The simulation results demonstrate that the proposed approach based on the concepts of the Derman–Lieberman–Ross theorem, outperforms other policies compared in our experiments and its superior performance is sustainable for inconsistent, processor-consistent, and task-processor-consistent types of heterogeneity.

5.2 Introduction

Distributed computing systems are widely used today for execution of large workloads composed of independent tasks that are divided among multiple heterogeneous compute nodes. The assignment of tasks to compute nodes is referred to in the literature as resource allocation or mapping. Effective mapping policies must account for multiple factors, e.g., the set of available compute nodes in the system, characteristics of these nodes, and links between them. Multiple scenarios can be identified where there is an uncertainty in the availability of functional compute nodes over time. Due to this uncertainty, any compute node may randomly fluctuate between the "failure" ("down") and "working" ("up") states.

First, consider a distributed system with a dynamic set of compute nodes, i.e., nodes may join and leave the system in an ad-hoc fashion. An example use of such a system is SETI at Home, composed of remote non-dedicated workstations that participate in distributed data processing [83]. Typically, compute nodes can go off-line anytime, regardless of the portion of the load assigned to them. Furthermore,

the participation of any node may be interrupted by local usage of the node by its owner.

A phenomenon known as "software aging" is a second example [38, 63]. Software aging sources include memory leaks, unreleased file locks, accumulation of unterminated threads, data corruption/round-off accrual, filespace fragmentation, and shared memory pool latching. Performance problems caused by software aging have become commonplace for computing resources including safety critical systems. For example, software aging of the Patriot Missile software was responsible for the loss of lives of American soldiers during the first Gulf War [67]. The solution found for this problem was to restart the Patriot software components every eight hours. Recovery mechanisms for software aging involve different, often proactive, fault management techniques for cleaning up system internal states to prevent the future occurrence of more severe failures or system performance degradation.

The malfunction of underlying hardware resources is another common source of changing availability of compute nodes in the system, due to harsh operating conditions or external physical impacts on the system. For example, broken cooling fans in a machine room typically cause a temperature increase that results in a high occurrence of processor malfunctions. Consequently, this initiates fatal errors or dramatic performance degradation that OS or process monitoring agents often resolve by restarting a process or rebooting an entire system.

The fault tolerant aspect of distributed computing systems has been extensively explored in the last few years [14]. Available literature on distributed computing in such uncertain environments primarily considers reactive techniques, where a node failure is addressed only after its occurrence. Checkpoint-resume or terminate-restart mechanisms are often used to recover unprocessed tasks at the failed nodes [22, 61].

Node failure also can be addressed by keeping multiple copies of the workload on different nodes [96]. These approaches are coupled in practice with redundancy schemes that duplicate system hardware resources entirely or partially. Depending on the implementation, duplicated resources are either always active or become active dynamically. Additionally, most of the existing literature that offers an analytical formulation of distributed-computing systems assumes a homogeneity among compute nodes and known system parameters [40, 84].

Clearly, the uncertainty in working compute nodes is expected to degrade the performance of any resource-allocation policy that does not account for node failure and recovery. In this study, we model a dedicated system composed of a limited set of heterogeneous Web servers. The availability of servers is described with exponential distributions with failure rates that are relatively high to simulate harsh operating conditions. Each HTTP request made by a user of the website is assumed to have a hard deadline, limiting the total time available to process it, and an associated reward. The goal is to design a resource-allocation policy that maximizes the expected cumulative reward received from the requests that finish before their deadlines, with the uncertainty of compute node failures.

The concept of robustness of a resource allocation was introduced in [6, 89], and later efforts applied the concept to resource management [4,89,93]. The mathematical model developed in this research does incorporate many of the same basic principles addressed in our earlier work on robustness. In analyzing systems where compute resources fail in a random fashion, the operational periods of the compute resources are uncertain. This uncertainty can impact the system by causing tasks to fail during execution and finally miss their deadlines. Often in practice, if the number of failed tasks reaches a certain level a service provider becomes subject to profit losses and

penalties. Therefore, a simplistic quantification for robustness can be defined as the difference between the actual (or expected) profit and the lowest level that justifies the operation of the system.

The major contribution of this paper is the design of a resource-allocation policy for the above environment based on the concepts of the Derman-Lieberman-Ross theorem [28]. Our simulation results demonstrate that the proposed solution outperforms other policies considered in this study. Its superior performance is sustainable in environments with different types of heterogeneity among tasks and compute nodes. Thus, the proposed resource-allocation mechanism can be applied to maximize the performance of a distributed heterogeneous system that experiences temporal compute node failures.

The remainder of this work is organized in the following manner. Section 5.3 presents the Derman-Lieberman-Ross theorem and a corollary used as a basis in this research. Section 5.4 describes an approximation scheme used to pose a simplified resource-allocation problem into the Derman-Lieberman-Ross framework. Section 5.5 builds on the solution for the simplified resource-allocation case by considering tasks with exponentially distributed execution times and different types of heterogeneity. Section 5.6 introduces the method developed to derive a distribution required for the Derman-Lieberman-Ross framework. Task deadlines and the technique used to estimate a probability that a task will be successfully completed through multiple reassignments are presented in Section 5.7. The parameters of the simulation setup are discussed in Section 5.8 along with the resource-allocation policies compared in this study. The simulation results are discussed in Section 5.9. A sampling of some related work is presented in Section 5.10. Section 5.11 concludes the paper.

5.3 DLR Policy

In 1972, Derman, Lieberman, and Ross introduced the following optimal policy for the sequential stochastic assignment problem [28]. Suppose there are N workers available to perform N jobs. The N jobs arrive in sequential order, i.e., job 1 arrives first, followed by job 2, etc. Associated with the jth ($j = 1, 2, ..., N$) job is a real-valued random variable X_j representing its worth. It will be assumed that the X_j are independent and identically distributed random variables with cumulative density function (cdf) $G_X(z)$ with finite mean value. If a "perfect" worker is assigned to the type jth job, a reward X_j is obtained. However, none of the N workers are perfect, and whenever the ith worker is assigned to the jth job, the reward is given by $p_i X_j$, where $0 \leq p_i \leq 1, i = 1, 2, ..., N$ represents the probability of worker i successfully completing any job. Each worker is assigned to one and only one job. The goal is then to assign the N workers to the N jobs so as to maximize the total expected reward. Let a policy be any rule for sequentially assigning workers to jobs. In particular, if $m(i)$ is defined to be the job to which ith worker is assigned, then the total expected reward is given by

$$E\left[\sum_{i=1}^{N} p_i X_{m(i)}\right]. \tag{5.1}$$

The desired policy is the one that maximizes this expected cumulative reward.

Intuitively, it appears reasonable to match a worth of a job to the probability of a worker of completing this job, e.g., assign an appeared high worth job to to a worker with high p_i. Quantifying this match requires $G_X(z)$ and values of p_i. The following Derman–Lieberman–Ross (DLR) theorem embodies this intuition providing the *optimal* resource-allocation policy using the available stochastic information (see [28] for further details).

DLR Theorem. *For each $N \geq 1$, there exist numbers $-\infty = a_{0,N} \leq a_{1,N} \leq a_{2,N} \leq \ldots \leq a_{N,N} = +\infty$, such that whenever there are N assignments to make and probabilities $p_1 \leq p_2 \leq \ldots \leq p_N$ then the optimal choice in the first assignment is to use the worker i such that X_1 is contained in the interval $(a_{i-1,N}, a_{i,N}]$. The $a_{i,N}$ depend on G_X but are independent of the p_i values and calculated recursively for N as follows:*

$$a_{i,N} = \int_{a_{i-1,N-1}}^{a_{i,N-1}} z\, dG_X(z) + a_{i-1,N-1} G_X(a_{i-1,N-1}) + a_{i,N-1}[1 - G_X(a_{i,N-1})],$$

with the convention that $a_{0,N} = -\infty$, $a_{N,N} = +\infty$, $-\infty \times 0 = 0$, and $\infty \times 0 = 0$.

Suppose that $N = 5$, and $G_X(z)$ is as illustrated in Fig. 5.1. When the DLR policy is applied, the $a_{i,5}$ values are calculated recursively starting from $N = 1$. These values divide the domain of all possible job worths into five intervals. Once the first job has arrived, a resource-management system identifies which of these five intervals this job falls in based on the job's worth X_1. Assume that this happened to be the third interval. Then, the resource-allocation system assigns the arrived job to the third worker in the sorted list of probability values $p_1 \leq p_2 \leq p_3 \leq p_4 \leq p_5$, as shown in Fig. 5.1. Sequentially as the next job arrives, N is decremented, and the same procedure is repeated by recalculating intervals.

5.4 Simplified Resource Allocation

Consider the following problem statement for a simplified resource allocation problem in a distributed system. Suppose there is a batch of N tasks. Each task i, $1 \leq i \leq N$, is characterized by: (a) the number of instructions it contains, denoted $\underline{n_i}$, and (b) reward $\underline{r_i}$ for completing this task. Suppose that N processors become available

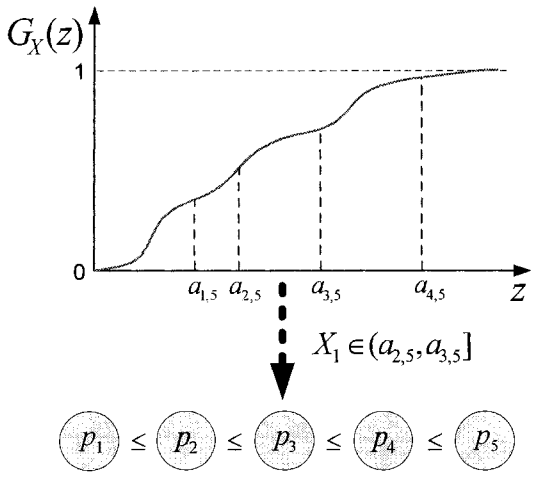


Figure 5.1: The distribution described with cdf $G_X(z)$ is divided into five intervals. According to the DLR policy, as the worth X_1 of the arrived job belongs to the third interval, this job will be assigned to the third worker in the list of workers ranked based on probability values p_i.

sequentially in time. Whenever a new processor becomes available, we need to assign one of the remaining (unassigned) tasks to this processor. Suppose that the jth processor to arrive is characterized by a random variable T_j that specifies the time that processor takes to execute each instruction (assuming that all instructions in a task take the same time to execute on the given processor). Moreover, the processor might randomly fail some time during the execution of a task, modeled as follows. The probability of failure is defined by its failure rate λ_j, so that the probability of failure during the execution of a task is given by λ_j multiplied by the execution time (we discuss this failure model further below).

Once task i is assigned to a processor, its execution begins, and reward r_i will be obtained if the task is completed successfully. If a processor fails while executing the task, no reward is earned and the task is removed from the batch, i.e., the task will never be executed again. Let $\mathbb{P}[i, m(i)]$ denote the probability that task i is

successfully completed when assigned to processor $m(i)$. Then, the goal is to maximize the total expected reward accumulated through N sequential assignments:

$$\text{maximize } E\left[\sum_{i=1}^{N} r_i \mathbb{P}[i, m(i)]\right]. \tag{5.2}$$

Using the failure model above, $\mathbb{P}[i, m(i)] = 1 - \lambda_{m(i)} n_i T_{m(i)}$. Substituting this into Eq. 5.2 and simplifying, the optimization problem becomes

$$\text{maximize } E\left[\sum_{i=1}^{N} \left([r_i n_i] \times \left[-\lambda_{m(i)} T_{m(i)}\right]\right)\right]. \tag{5.3}$$

The set of workers and arriving jobs in the sequential stochastic assignment problem considered in Section 5.3 correspond to tasks in the batch and sequentially arriving processors considered here, respectively. Moreover, the performance goal given by Eq. 5.3 is structured identically to Eq. 5.1. Therefore, if we assume that the random variables $X_j = \lambda_j T_j$, $j = 1, \ldots, N$, have known cdf $G_X(x)$ with a finite mean, then the policy derived in the DLR theorem can be applied. We can think of X_j as a basic failure parameter associated with processor j, which has the interpretation of *per-instruction failure rate.*

The failure model above can be viewed as an approximation to the standard model where the time it takes for failure is exponentially distributed with parameter λ_j. In this case, the probability that processor $m(i)$ fails while executing task i is given by $e^{-\lambda_{m(i)} n_i T_{m(i)}}$. Using the Taylor series expansion for the exponential function,

$$e^{-\lambda_{m(i)} n_i T_{m(i)}} = 1 - \lambda_{m(i)} n_i T_{m(i)} + \frac{(-\lambda_{m(i)} n_i T_{m(i)})^2}{2!} + \frac{(-\lambda_{m(i)} n_i T_{m(i)})^3}{3!} + \cdots.$$

Ignoring second order and higher terms, we arrive at the failure model used in the

derivation above.

5.5 Tasks with Exponentially Distributed Execution Times

In the simplified resource-allocation case described in the previous section, observe that (1) if tasks are ordered according to their $r_i n_i$ values this order does not depend on processors j, and (2) the objective function for the resource-allocation problem is based on the product $[r_i n_i] \times \left[-\lambda_{m(i)} T_{m(i)}\right]$. Fig. 5.2 illustrates this relationship between tasks and processors in a matrix form as the Cartesian product of two sequences: $r_i n_i$, $1 \leq i \leq N$ and $-\lambda_j T_j$, $1 \leq j \leq N$. Note that the matrix is ordered in both row and column directions. The DLR theorem provides the optimal assignment policy for such matrices given that processor's characteristic $-\lambda_j T_j$ is a random variable with a known distribution.

	$-\lambda_1 T_1$	$\leq$	$-\lambda_2 T_2$	$\cdots$	$\leq$	$-\lambda_N T_N$
$r_1 n_1$	$-\lambda_1 T_1 r_1 n_1$	$\leq$	$-\lambda_2 T_2 r_1 n_1$	$\cdots$	$\leq$	$-\lambda_N T_N r_1 n_1$
$\leq$	$\leq$		$\leq$			$\leq$
$r_2 n_2$	$-\lambda_1 T_1 r_2 n_2$	$\leq$	$-\lambda_2 T_2 r_2 n_2$	$\cdots$	$\leq$	$-\lambda_N T_N r_2 n_2$
$\vdots$	$\vdots$		$\vdots$			$\vdots$
$\leq$	$\leq$		$\leq$			$\leq$
$r_N n_N$	$-\lambda_1 T_1 r_N n_N$	$\leq$	$-\lambda_2 T_2 r_N n_N$	$\cdots$	$\leq$	$-\lambda_N T_N r_N n_N$

Figure 5.2: An example matrix based on the Cartesian product for the simplified resource-allocation example.

In the literature on distributed systems, Estimated Time to Compute (ETC) matrices often are assumed to be given [49, 57, 99]. Each entry in an ETC matrix is an estimate of time, t_{ij}, to compute task i on processor j. Clearly, for the simplified

resource-allocation problem presented in Section 5.4, the ETC matrix would be based on a Cartesian product between n_i and T_j sequences because $t_{ij} = n_i T_j$. However, in practice, it is difficult or even impossible to represent t_{ij} with such a product. This is mainly because the exact number of executed instructions is usually unknown in advance due to conditional branching, uncertain input data, etc. Furthermore, instruction execution rates of participating processors may depend on the mix of the types of executed instructions [46]. Therefore, an ETC matrix based on empirical measurements typically provides a more accurate means of capturing the relationship among tasks and processors in distributed systems.

The rest of this paper will focus on tasks with execution times exponentially distributed for each task-processor pair. This assumption implies that an ETC matrix contains expected execution times that are assumed to be known (expected values, i.e., means, are sufficient to fully describe exponential distributions [29]). Eq. 5.3 can be restructured as:

$$\text{maximize } E\left[\sum_{i=1}^{N}\left(r_i \times \left[-\lambda_{m(i)} t_{im(i)}\right]\right)\right]. \tag{5.4}$$

In this study, we distinguish between three major classes of heterogeneity in distributed systems: task-processor-consistent, processor-consistent, and inconsistent. The first class of heterogeneity describes systems where two conditions are satisfied: (1) if task A requires more time to execute than task B on one processor then the same is true for any other processor in the system; (2) if processor A requires more time to execute one task than processor B then it is true for any other task in the batch. The second type of heterogeneity implies condition (2) only. The third class describes systems where neither of these two conditions are met.

Fig. 5.2 demonstrates a matrix where all rows and columns are ranked in ascending order. This may not be the case for many ETC matrices where t_{ij} entries are not represented by products of two independent components. As a result, the processors may not be ranked identically for each row. To satisfy the "global" ordering required to apply the DLR framework, the average mean execution values t_j^{av}, $1 \leq j \leq N$, used in this work are computed for each processor j across all the tasks left unassigned in the batch, i.e., $t_j^{av} = \sum_{i=1}^{N} t_{ij}/N$. Because the set of such tasks changes as a resource-allocation process continues, the t_j^{av} values change as well, which may change the ordering of processors.

Once the processors and tasks are sorted in ascending order of $-\lambda_j t_j^{av}$ and r_i rewards, respectively, matrix entry products $-\lambda_j t_j^{av} r_i$ in some rows and columns might not completely follow these "global" orders as shown in Fig. 5.3. To evaluate qualitatively how much, on average, a given ETC matrix deviates from each of these "global" orders, Δ_r and Δ_c metrics are introduced for rows and columns, respectively. Consider the Δ_r metric. First, the Bubble sort algorithm [58] is hypothetically applied to each row, and the number of swaps required to match the corresponding "global" order for processors is counted. Then, Δ_r is calculated as the average number of swaps across all the rows in the matrix. Similarly, the Δ_c metric is computed for the columns of the original matrix.

5.6 Distribution of Processors

The DLR theorem assumes that the distribution describing the relative random availability of each processor is known and constant over time. In the "Internet computing" model [22], where processors arrive from a large pool of external users, the required

	$-\lambda_1 t_1^{av}$	$\leq$	$-\lambda_2 t_2^{av}$	$\leq$	$-\lambda_3 t_3^{av}$
r_1	$-\lambda_1 t_{11} r_1$	$\leq$	$-\lambda_2 t_{12} r_1$	$\leq$	$-\lambda_3 t_{13} r_1$
$\leq$	$\geq$		$\leq$		$\leq$
r_2	$-\lambda_1 t_{21} r_2$	$\leq$	$-\lambda_2 t_{22} r_2$	$\geq$	$-\lambda_3 t_{23} r_2$
$\leq$	$\leq$		$\leq$		$\leq$
r_3	$-\lambda_1 t_{31} r_3$	$\leq$	$-\lambda_2 t_{32} r_3$	$\leq$	$-\lambda_3 t_{33} r_3$

Figure 5.3: The example matrix demonstrates that the orders established for processors and tasks required in the DLR framework are not followed completely for the first processor and for the second task.

distribution can be constructed based on the past history of processor arrivals. In contrast, distributions in dedicated distributed systems need to be derived based on characteristics of the tasks batched and the parameters of the limited set of available processors.

Fig. 5.4 illustrates a probability mass function (pmf) for a dedicated system composed of M processors. The values $-\lambda_k t_k^{av}$, $1 \leq k \leq M$, arranged in ascending order, can be used to compute the "bin boundaries" $a_{i,N}$ according to the DLR policy. Let p_k, $1 \leq k \leq M$, denote a probability associated with each processor, characterized

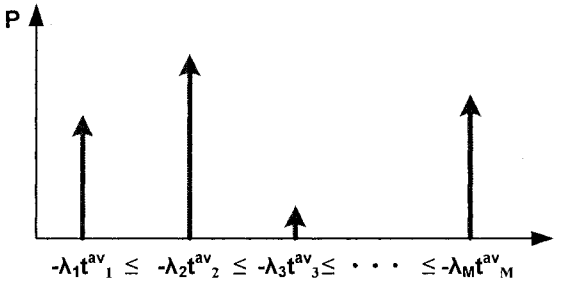


Figure 5.4: The general pmf for a dedicated system with M processors.

by $-\lambda_k t_k^{av}$, in the pmf shown in Fig. 5.4. To evaluate the p_k values, consider the following models describing the availability of processors in a dedicated distributed

system. The first two models are of little practical value; they are presented here solely to provide a transition to the third model actually used in this study.

Identical failure rates, processor becomes available just after failure: Assume that each processor has the same failure rate λ and becomes available for the next assignment immediately after a failure. Assume also that task completions do not make a processor available. Clearly in this situation, each processor fails, on average, the same number of times, i.e., probabilistic components p_k are the same in the pmf shown in Fig. 5.4 can be computed as $p_k = 1/M$.

Individual failure rates, processor becomes available just after failure: When each processor has its own failure rate λ_k, it fails over time, on average, every λ_k^{-1} time units. Therefore, the probability components p_k can be calculated as $p_k = \lambda_k / \sum_{k=1}^{M} \lambda_k$.

Individual failure rate, processor available just after failure and task completion: Suppose that each processor has failure rate λ_k and becomes available immediately after a failure or a task completion. Let $\underline{w_k}$ denote the total availability rate for processor k. Assuming that times between failures and task completions are exponentially distributed for each processor, w_k can be computed as:

$$w_k = \lambda_k + \frac{1 - e^{-\lambda_k t_k^{av}}}{t_k^{av}}. \tag{5.5}$$

The second term in the sum in Eq. 5.5 estimates the processor's availability rate from task completions. On average, processor k would complete a task every t_k^{av} if there were no failures. The actual availability rate from task completions is lower than that because the probability of having no failures during t_k^{av} is accounted for in $1 - e^{-\lambda_k t_k^{av}}$. As before, the probability components p_k can be calculated by normalizing the w_k

values, i.e., $p_k = w_k / \sum_{k=1}^{M} w_k$.

Based on our simulation studies, the following iterative scheme to compute the pmf was found to be the most efficient. According to this scheme, the distribution, described with a weighted sum of two pmfs, is recalculated at each step of the mapping process before the assignment of the next task takes place. The first pmf is a normalized histogram constructed from the actual number of times that each processor has become available since the execution of the batch started. For the second pmf, the processor availability rates w_j are recomputed because the number of tasks left in the batch decreases with time. Let $N(t_0)$ denote the initial number of tasks in the batch and let $N(t)$ denote the number of tasks left in the batch at time t. Then, $N(t)N(t_0)^{-1}$ and $1 - N(t)N(t_0)^{-1}$ are the weighting factors for the first and second pmfs in the sum, respectively. As such, these weighting factors are adjusted at each iteration proportionally to the progress made.

5.7 Tasks with Deadlines

Suppose that in addition to the exponentially distributed execution time, each task has a deadline d_i, i.e., the reward r_i will be earned only if task i is successfully completed by time d_i. This means that if a processor fails before d_i while executing task i then task i returns to the batch, so it can be reassigned again. When d_i expires, the task is dismissed from the batch. Fig. 5.5 illustrates the possible states for a task and transitions between these states assuming that the considered task eventually fails.

To account for possible multiple reassignments of each task in the resource-allocation

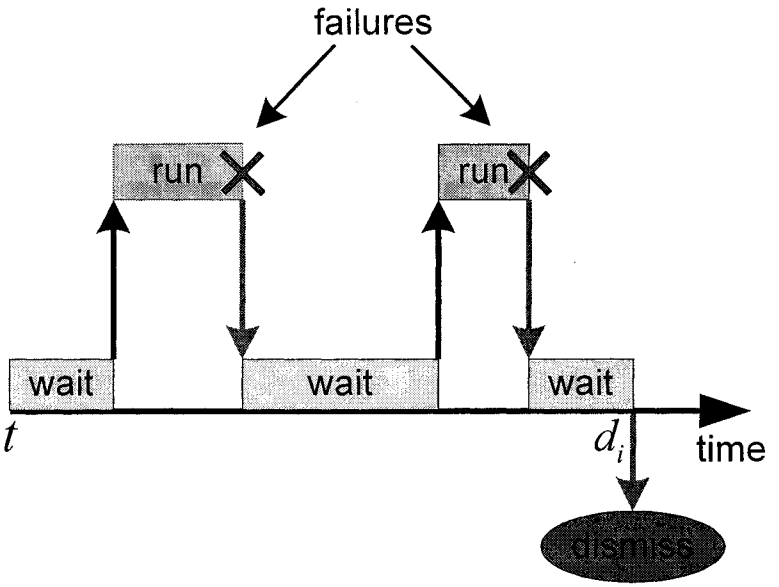


Figure 5.5: Any task in the batch waits in the queue before it is assigned and returns back to the queue if a processor fails during its execution. If d_i expires the task is dismissed from the queue.

policy, let $\underline{\mathbb{P}_i(t)}$ be an estimate of the probability at time t that task i will be successfully completed before its deadline d_i. Then, Eq. 5.4 can be adapted as follows:

$$\text{maximize } E\left[\sum_{i=1}^{N(t)}\left[r_i\mathbb{P}_i(t)\right]\left[-\lambda_{m(i)}t_{im(i)}\right]\right]. \tag{5.6}$$

The following approach was used in this study to estimate $\mathbb{P}_i(t)$. Let $\underline{V_i(t)}$ denote the expected number of reassignments for task i from time t until its deadline. As the probability of failure for the assigned task i is:

$$\sum_{j=1}^{M} p_j(1-e^{-\lambda_j t_{ij}}), \tag{5.7}$$

the probability $\mathbb{P}_i(t)$ follows:

$$\mathbb{P}_i(t) = 1 - \left(\sum_{j=1}^{M} p_j (1 - e^{-\lambda_j t_{ij}}) \right)^{V_i(t)}. \tag{5.8}$$

Let $T_i^{wait}(t)$ denote how long, on average, task i spends in a "wait" state (see Fig. 5.5). Similarly, let T^{run} denote the average time for a task to spend in a "run" state. Once $T_i^{wait}(t)$ and T^{run} are found, $V_i(t)$ can be computed as:

$$V_i(t) = \frac{d_i - t}{T_i^{wait}(t) + T^{run}}. \tag{5.9}$$

Assuming that any task from the batch can be assigned to any processor, the average expected delay from the time when a task is assigned to a processor to the time when the processor fails can be computed as the mean across λ_j^{-1} values:

$$T^{run} = \sum_{j=1}^{M} p_j \lambda_j^{-1}. \tag{5.10}$$

A delay expected between sequential assignments for a given task, i.e., $T_i^{wait}(t)$, depends on (1) how often processors become available and (2) how many tasks remain, on average, in the batch. To estimate (1), let W denote the overall processor availability rate in the system. As shown in Section 5.6, availability rate w_j for processor j can be estimated with Eq. 5.5. Similarly, in the system composed of M processors, W can be computed as a sum of w_j rates:

$$W = \sum_{j=1}^{M} w_j. \tag{5.11}$$

Although task completions and deadline expirations change the number of tasks that

remain in the batch, in this study we use a lower bound for this number assuming that only the tasks with expired deadlines are dismissed over the time period $d_i - t$. This results in the following estimate for $T_i^{wait}(t)$:

$$T_i^{wait}(t) = \frac{N(t) + N(d_i)}{2 \times 2W}. \tag{5.12}$$

In Eq. 5.12, $\frac{N(t)+N(d_i)}{2}$ gives the average length of the queue over the time period $d_i - t$. This length is divided by 2 assuming that, on average, 50% of the remaining tasks will be assigned before task i. Our extensive experiments show that even if the absolute values found for $\mathbb{P}_i(t)$ with the described method are not quite accurate, they result in a good relative ranking among the tasks—the only factor that matters in the DLR framework.

5.8 Simulation Setup and Resource-allocation Policies

The prototype system studied in this research is a dedicated heterogeneous computer cluster processing HTTP requests (tasks). Typically, such systems belong to a more general class of heterogeneous computing systems where new tasks arrive dynamically into the system, and their arrival times are, usually, not known in advance. In this study, we address a simplified case and work with a static environment assuming that a batch is filled with HTTP requests (i.e., tasks) without considering any new arrivals during a given mapping event.

To generate ETC matrices to be used in the simulations, we assume that each task in the batch belongs to one of five classes, based on its complexity. Each task

class is defined by a set of exponential distributions, where each distribution describes the probability of all execution times for that class on a given processor within the heterogeneous suite. To specify each distribution, the mean execution time is generated randomly in the range of $[0.5, 4]$ sec. for each task-class processor pair. As a result of this random approach, an unsorted ETC matrix models the inconsistent heterogeneity described in Section 5.5. If the elements of each row in this matrix are independently sorted in ascending order, the matrix models the processor-consistent heterogeneity. If the elements of each column in the matrix are independently sorted in ascending order, that matrix models the task-processor-consistent heterogeneity. Note that as a new ETC matrix is created after each sorting procedure it represents a completely different distributed system.

A batch for each simulation trial consisted of 200 tasks. Each task was randomly associated with one of the five classes. Each HTTP request (i.e., task) made by a user of the website must be completed within a deadline d_i. If a task cannot be completed by its deadline then the request is considered timed out and will be discarded from the batch. The deadline for each task was set at a certain level $\underline{D}$ based on the longest mean execution time for that task class determined across all the processors. The performance of resource-allocation policies was explored with respect to two proportionality factors D: 300% and 600%.

Integer task rewards r_i were generated in two different ways. The *execution time independent rewards* method computed the reward r_i for each task i by sampling a uniform distribution in the range of $(0, 100]$ regardless of task execution times t_{ij}. In contrast, the *execution time dependent rewards* method generated reward r_i based on a task class. Specifically, the interval $(0, 100]$ was divided evenly into five subintervals, and r_i was determined by sampling a uniform distribution on a

corresponding subinterval. For example, if task i belonged to the third class its reward value r_i was generated from subinterval $(40, 60]$. The second method appears to be plausible in many practical applications as it implies that tasks with longer execution times are more valuable in task-processor consistent systems.

Typically, hardware failure rates are very low when a distributed system operates under normal operating conditions [104]. However because the goal of our research is to maximize system performance under harsh operating conditions, the simulated mean time between failures was set for each processor relatively high, i.e., within the range of $[0.6, 1]$. As mentioned before, a processor in the simulation model becomes available either just after a hardware failure or just after a task completion. Once a task is assigned to a processor it is removed from the batch; if it fails it is returned to the batch. A task is discarded when its deadline expires.

The performance of the following five different resource-allocation policies was explored in the given environment.

1. In the **deadline** policy, a task with the closest deadline d_i is assigned to the next available processor.

2. In the **reward** policy, a task with the highest reward r_i is assigned to the next available processor.

3. In the **reward_P** policy, a task with the highest value $r_i\mathbb{P}_i(t)$ is assigned to the next available processor, where $\mathbb{P}_i(t)$ is computed as described in Section 5.7

4. In the **DLR** policy, once a processor becomes available, a resource-allocation system takes the following steps to select the next task for assignment:

   (a) sorts the remaining $N(t)$ tasks in ascending order of r_i values;

(b) recomputes the distribution of processors, as explained in Section 5.6;

(c) calculates "bin boundaries," for $N(t)$ according to the DLR theorem;

(d) determines the index i of the "bin" that the arrived processor corresponds to;

(e) selects task with index i for assignment from the list of tasks formed in step (a).

5. In the **DLR_P** policy, the steps are the same as in the DLR policy, but the task list is sorted based on $r_i \mathbb{P}_i(t)$ values.

5.9 Experimental Results

The resource-allocation policies described above were compared in the simulated environment based on their ability to maximize the cumulative reward, averaged over 100 simulation trials. For each of these trials performed for the same set of parameters (i.e., distributions, number of tasks, etc.), a random number generator was seeded differently. This allowed the performance of the resource-allocation policies to be explored over a broad range of samples pulled from the corresponding distributions. In each simulation trial, the ETC matrix was generated in a random fashion to model inconsistent heterogeneity. The same matrix was sorted in the row direction and, then, in the column direction for the processor-consistent and task-processor-consistent scenarios, respectively. For each ETC matrix used in the experiment, the Δ_r and Δ_c metrics were computed for **DLR** and **DLR_P** policies, as described in Section 5.5, to estimate how well the matrix complies with the ordering requirements of the DLR theorem.

The average performance across 100 simulation trials along with the corresponding 95% confidence intervals are presented in Fig. 5.7 for all five resource-allocation policies. The **DLR_P** consistently delivers the best results for all three types of heterogeneity explored in this study. The superior performance of this policy is based on two factors incorporated into the decision making process: (1) the stochastic information describing the availability of processor types in the system, and (2) the probability $\mathbb{P}_i(t)$ estimated for task i to be successfully completed through multiple reassignments. The second factor plays a very important role: once integrated into the **reward_P** policy, it improves its performance by 22%, on average, making it comparable the performance of the **DLR** policy.

Fig. 5.6 demonstrates the progress of each policy over time in one of the simulation trials for processor-consistent heterogeneity with execution time independent rewards. This trial was selected because it reflects the average performance results plotted in Fig. 5.7(a). As expected, the **reward** and **reward_P** policies quickly accumulate the total reward in the beginning as they select the most profitable tasks first. When the number of such tasks becomes smaller, the total reward accumulation slows down as a result of more frequent task failures. Furthermore, closer to the end, the **reward** policy experiences a significant loss of tasks due to expired task deadlines.

Tables 5.1 and 5.2, show the total number of successfully completed tasks averaged over 100 simulation trials. Despite the fact that the **deadline** policy completes the highest number of tasks, its performance remains the worst because many of the tasks have low reward values. In contrast, the DLR-based policies have rather moderate numbers of completed tasks, but the demonstrated performance results highlight their ability of selecting tasks on a "more intelligent" basis to maximize the cumulative reward.

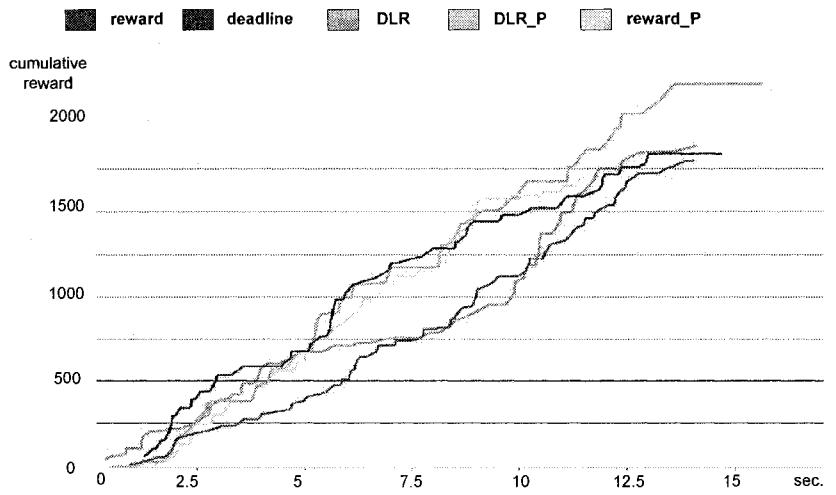
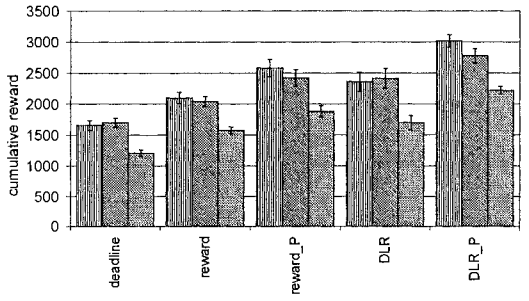


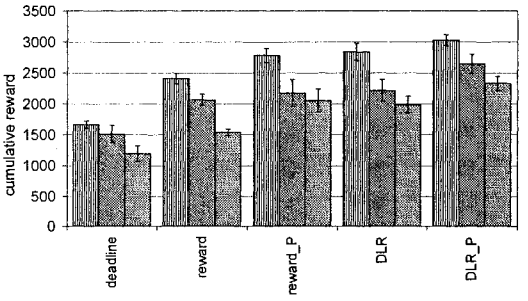
Figure 5.6: A progress over time of the five resource-allocation policies with respect to the cumulative reward captured in one simulation trial for processor-consistent heterogeneity.

Table 5.1: Average numbers of successfully completed tasks computed across 100 simulation trials. Execution time independent rewards.

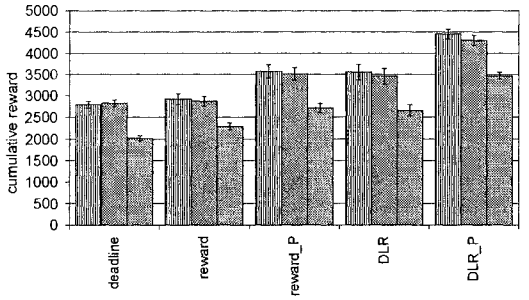
policy	D	heterogeneity		
		task-processor consistent	processor consistent	inconsistent
deadline	300%	48	47	44
reward	300%	33	35	29
reward_P	300%	37	31	27
DLR	300%	41	37	32
DLR_P	300%	43	38	33
deadline	600%	75	73	61
reward	600%	64	57	53
reward_P	600%	63	58	50
DLR	600%	66	69	51
DLR_P	600%	67	67	53



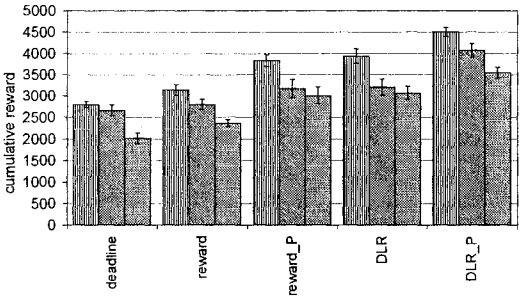
(a) $D = 300\%$; ex. time independent rewards



(b) $D = 300\%$; ex. time dependent rewards

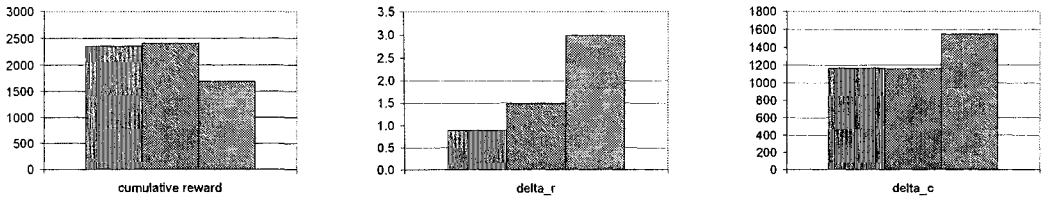


(c) $D = 600\%$; ex. time independent rewards

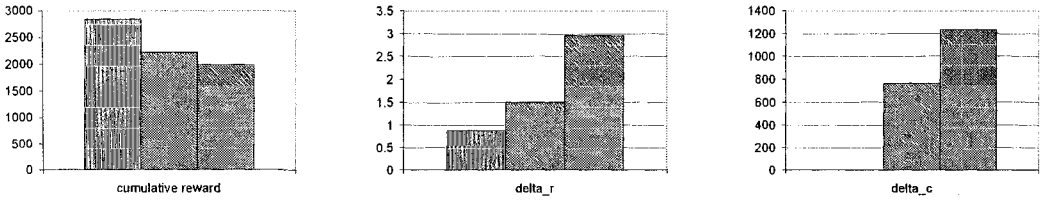


(d) $D = 600\%$; ex. time dependent rewards

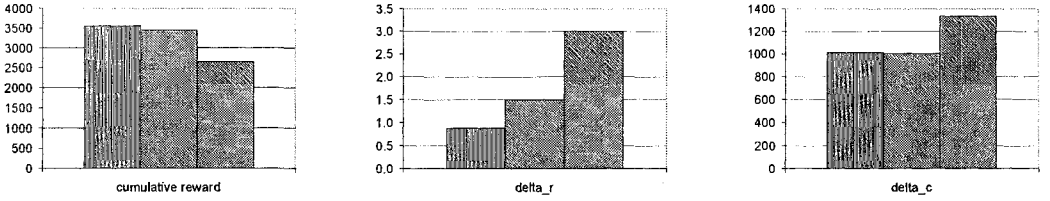
Figure 5.7: The performance of five resource-allocation policies explored over the two methods of assigning rewards r_i and two setups for D.



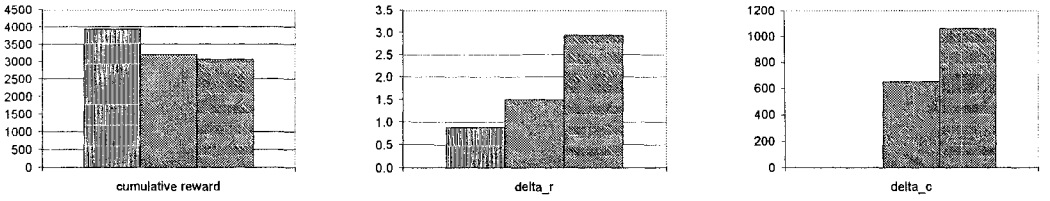
$D = 300\,\%$; ex. time independent rewards



$D = 300\,\%$; ex. time dependent rewards



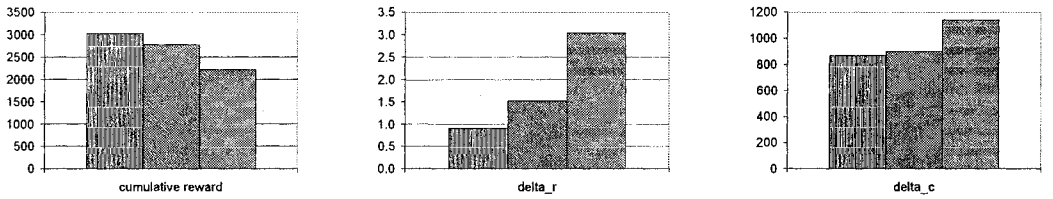
$D = 600\,\%$; ex. time independent rewards



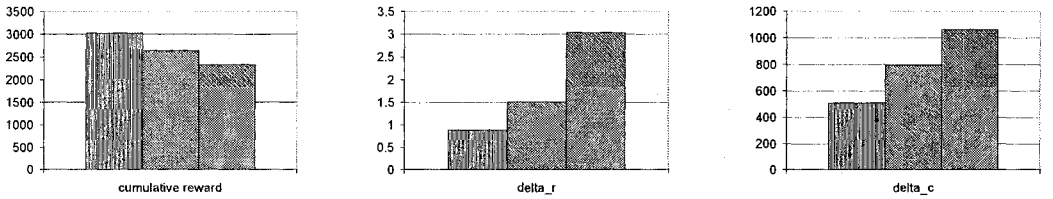
$D = 600\,\%$; ex. time dependent rewards

task-processor consistent procesor consistent inconsistent

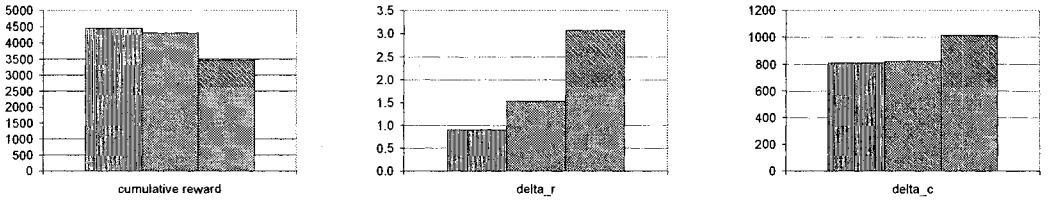
Figure 5.8: **DLR** policy: correlation between cumulative performance, Δ_r, and Δ_c.



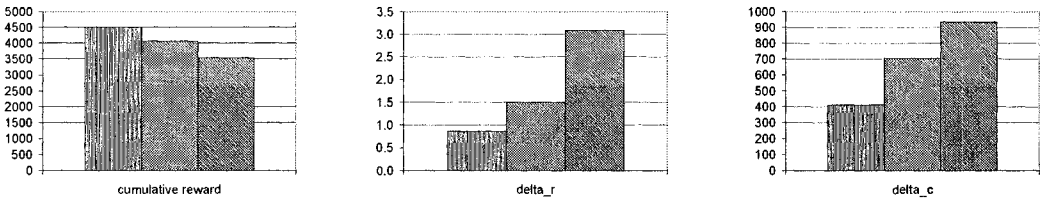
D = 300 %; ex. time independent rewards



D = 300 %; ex. time dependent rewards



D = 600 %; ex. time independent rewards



D = 600 %; ex. time dependent rewards

Figure 5.9: **DLR_P** policy: correlation between cumulative performance, Δ_r, and Δ_c.

Table 5.2: Average numbers of successfully completed tasks computed across 100 simulation trials. Execution time dependent rewards.

policy	D	heterogeneity		
		task-processor consistent	processor consistent	inconsistent
deadline	300%	48	47	28
reward	300%	42	37	26
reward_P	300%	39	35	24
DLR	300%	43	43	26
DLR_P	300%	43	41	25
deadline	600%	77	73	61
reward	600%	64	50	43
reward_P	600%	63	60	51
DLR	600%	66	65	54
DLR_P	600%	68	58	50

It is easy to observe that the average results shown in Fig. 5.8 and Fig. 5.9 demonstrate a correlation between the reduction in Δ_r and Δ_c values and the performance improvement of the DLR-based policies. In both cases, a significant transition occurs from the inconsistent to the processor-consistent type of heterogeneity. This can be explained by the fact that t_j^{av} values dominate in $-\lambda_j t_j^{av}$ expressions, which are used to rank processor types as described in Section 5.5. Thus, the majority of rows in the ETC matrix for the processor-consistent type of heterogeneity will be ranked in the same manner as the processor types in the DLR framework.

A task-processor consistent heterogeneity in the **DLR** policy, does not result in an additional performance improvement with with *execution time independent* rewards because only r_i values are used to rank tasks. The slight improvement for this reward generation method is observed in the **DLR_P** policy due to the fact that t_{ij} values are involved in the method of computing $\mathbb{P}_i(t)$, and, consequently, $r_i\mathbb{P}_i(t)$ are used to characterize tasks in that policy. However, when the reward generation method is changed to *execution time dependent*, a significant performance improvement is

obtained in the task-processor consistent case (see Fig. 5.7.) This can be explained by the fact that all the columns in the ETC matrix were ranked exactly as required in the DLR framework, i.e., $\Delta_c = 0$ (see Fig. 5.8.)

5.10 Related Work

The original work of Derman et al. [28] inspired research in various areas. Example applications of the DLR theorem include selling houses and job-search strategy [1]. In 1972, Albright and Derman determined the limiting behavior of the $a_{i,N}$'s as N becomes large [3]. The derived closed form solution can be used to avoid lengthy computation of $a_{i,N}$'s in such cases. Later, these authors addressed cases where the arrival process is a non-homogeneous Poisson process with a general discount function [2]. Nakai permits the distribution of resources to change according to a partially observable Markov process and allows the number of resources to be random [72]. Sakaguchi allows a fixed time horizon [81] and permits resource values to be dependent [82]. The results of these studies were applied to investment strategies [80], firing torpedoes at randomly arriving targets [82], allocating organs for transplants [26], and manufacturing and telecommunications [79]. Similar to our work, all aforementioned studies address a sequential assignment problem in different in problem domains, i.e., at each time when the resource-allocation system observes a realization of random variable, it must select the best action, one at a time in sequential order.

As mentioned before, the fault tolerant aspect of modern distributed computing systems has been extensively explored. However, the available literature on distributed computing in such uncertain environments primarily considers reactive techniques, where a node failure is addressed only after its occurrence [14]. One of the

few exceptions is the paper of Dhakal et al. [30] that presents two preemptive load-balancing policies for a heterogeneous distributed computing system with wireless links between nodes. Preemptiveness in this case implies adjusting actions to compensate for the possibility of node failure/recovery. The main goal for these policies is to avoid a scenario where a node fails while having a large amount of unprocessed load. The data transfer of such load to other nodes may result in a large random delay over wireless channels with following idle times on other nodes. A probabilistic model, based on the concept of regenerative processes, is presented to assess the overall performance of the system under these policies. The experiments show that preemptively utilizing the statistical information about the failure and recovery processes to adjust the load-balancing gain to an optimal value, allows one to minimize the mean of the overall completion time of the total workload. Although the problem domain differs from ours, [30] exemplifies an effective integration of the available node failure/recovery statistics into the resource-allocation process.

5.11 Conclusion

This paper presents a method for robust static resource allocation in distributed systems under high failure rate. Given a batch of tasks, a resource-allocation system assigns tasks to compute resources that become available just after recovery from failures or task completions. The major contribution is the design of a resource-allocation policy for the above environment based on the concepts of the Derman-Lieberman-Ross theorem. The derived policy maximizes the expected cumulative reward received from the tasks that finish before their deadlines, given that the compute resources fail in a random fashion.

The resource-allocation policy was derived for the case where tasks and processors are categorized by the number of instructions and the time required to execute one instruction, respectively. Further, this policy was adapted to accommodate ETC matrices that provide a more accurate means of capturing the relationship among tasks and processors in distributed systems. As this study focused on dedicated distributed systems with a *limited* set of compute resources, the distribution describing the relative random availability of each processor was derived in Section 5.6 based on the characteristics of the tasks batched and the parameters of the processors available in the system.

The superior performance demonstrated by the **DLR** resource-allocation policy reveals its great potential for a variety of applications in a broad spectrum of distributed systems. For example, the problem of maximizing the performance when the system experiences temporal failures of compute resources is an important issue for embedded systems (e.g., [75, 91]), sensor networks (e.g., [108]), or special purpose cluster-based systems (e.g., [89]). Similar to the environment considered in this work, such systems sometimes are employed under harsh conditions but must deliver a certain level of performance to remain in operation. The application domains include surveillance for homeland security, monitoring vital signs of medical patients, and automatic target recognition systems. Thus, the proposed DLR-based resource-allocation scheme can be adapted in those systems.

Chapter 6

Conclusions

The robust design of computing and communication systems is becoming an increasingly important issue. This research addressed the problem of defining a robustness metric and developing effective resource-allocation techniques in various environments.

The first chapter presented a hybrid two-stage approach to initial resource allocation in the ARMS shipboard computing system. It included the design of a static combinatorial heuristic for a set of application strings that must be assigned to a set of heterogeneous machines connected by a high-speed network. The application strings are fed by a continuous stream of data sets arriving from sensors, and the processing system has to satisfy a certain QoS level of throughput and latency. A model of the application strings and underlying hardware platform was used to quantify a metric for robustness against unpredictable increases of the workload, i.e., the load presented by the set of sensors. Then, the established metric was maximized with an evolutionary Genitor-based search method coupled with a specially designed string allocation routine. The solution was passed to the second stage where the Branch-and-Bound

algorithm improved it and tightened the upper bound.

The second chapter addressed the problem of workload allocation in the IBM cluster-based printing system. In this system, an input stream of data, described using the Postscript language, arrives for rasterization. Requests for rasterization are processed by a dedicated cluster of workstations, where individual requests are distributed to the cluster by a centralized dispatcher. The collection of processed 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 today's fastest workstations. The primary contributions of this study were a mathematical model of a dynamic distributed computing system with hard deadlines on task execution times and an application of this model to the design of a resource allocation heuristic suitable for this type of system.

In the first part of the next chapter, a new *stochastic* robustness metric was presented where the uncertainty in system parameters and its impact on system performance were modeled stochastically. This stochastic model was then used to derive a quantitative evaluation of the robustness of a given resource allocation as the probability that the resource allocation will satisfy the expressed QoS constraints. In the second part of this chapter, the proposed method of stochastic robustness evaluation was integrated into greedy and global search heuristics developed to address the problem of resource allocation for a class of distributed systems operating on periodic data sets. In many systems of the considered class, it is highly desirable to minimize the period between subsequent data arrivals while providing a probabilistic guarantee that each data set is processed within a limited time.

The last chapter presented a method for robust static resource allocation in distributed systems under high failure rate. Given a batch of tasks, a resource management system assigns tasks to compute resources that become available just after recovery from failures or task completions. The major contribution was the design of a resource-allocation policy for the above environment based on the concepts of the Derman-Lieberman-Ross theorem. The derived policy maximizes the expected cumulative reward received from the tasks that finish before their deadlines, given that the compute resources fail in a random fashion. The superior performance demonstrated by the developed **DLR_P** resource-allocation policy reveals its great potential for a variety of applications in a broad spectrum of distributed systems. For example, the problem of maximizing the performance when the system experiences temporal failures of compute resources is an important issue for embedded systems, sensor networks, and a variety of military applications.

Vita

Vladimir Shestak is pursuing a Ph.D. degree from the Department of Electrical and Computer Engineering at Colorado State University, where he has been a Research Assistant since August 2003. He received his M.S. degree in Computer Engineering from New Jersey Institute of Technology in 2003 and his B.S. degree in Electrical Engineering from Moscow Engineering Physics Institute in 1998. From 1998 to 2002, he worked as a Network Engineer for CISCO Business Unit in Moscow, Russia. His research interests include resource management within distributed computing systems, algorithm parallelization, image processing, and computer network design and optimization. He currently works for InfoPrint Solution Company, Boulder, CO (former IBM Printing Systems Division).

Bibliography

[1] S. C. Albright, "Optimal sequential assignments with random arrival times," *Management Science*, vol. 21, no. 1, pp. 60–67, Sep. 1974.

[2] ———, "A Bayesian approach to a generalized house selling problem," *Management Science*, vol. 24, no. 4, pp. 432–440, Dec. 1977.

[3] S. C. Albright and C. Derman, "Asymptotic optimal policies for the stochastic sequential assignment problem," *Management Science*, vol. 19, no. 1, pp. 46–51, Sep. 1972.

[4] 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, *Parallel, Distributed, and Pervasive Computing*, ser. Advances in Computers. Amsterdam, The Netherlands: Elsevier, 2005, pp. 91–128.

[5] 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. 4, Dec. 2002.

[6] 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.

[7] S. Ali, J.-K. Kim, H. J. Siegel, and A. A. Maciejewski, "Static heuristics for robust resource allocation of continuously executing applications," *Journal of Parallel and Distributed Computing*, 2008.

[8] 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.

[9] Adaptive and Reflective Middleware Systems (ARMS). Accessed Dec. 10, 2005. [Online]. Available: http://dtsn.darpa.mil/ixo/ixo_FeatureDetail.asp?id=6#

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

[11] 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.

[12] G. Bernat, A. Colin, and S. M. Peters, "WCET analysis of probabilistic hard real-time systems," *23rd IEEE Real-Time Systems Symposium (RTSS '02)*, 2002.

[13] N. Bharadwaj and V. Chandrasekar, "Waveform design for CASA X-band radars," *32nd Conference on Radar Meteorology of American Meteorology Society*, Oct. 2005.

[14] K. Birman, *Reliable Distributed Systems.* New York, NY: Springer-Verlag, 2005.

[15] L. Bölöni and D. Marinescu, "Robust scheduling of metaprograms," *Journal of Scheduling*, vol. 5, no. 5, pp. 395–412, Sep. 2002.

[16] 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.

[17] T. D. Braun, H. J. Siegel, and A. A. Maciejewski, "Static mapping heuristics for tasks with dependencies, priorities, deadlines, and multiple versions in heterogeneous environments," *16th International Parallel and Distributed Processing Symposium (IPDPS'02)*, Apr. 2002, pp. 78–85.

[18] Casa. Accessed Mar. 06, 2007. [Online]. Available: http://www.radar.colostate.edu/casa/

[19] S. Chaudhuri, R. A. Walker, and J. E. Mitchel, "Analyzing and exploiting the structure of the constraints in the ILP approach to the scheduling problem," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 2, no. 4, pp. 152–164, Dec. 1994.

[20] E. K. P. Chong and S. H. Zak, *An Introduction to Optimization*, 2nd ed. New York, NY: John Wiley, 2001.

[21] E. G. Coffman, Ed., *Computer and Job-Shop Scheduling Theory.* New York, NY: John Wiley & Sons, 1976.

[22] Condor: High throughput computing. Accessed March 31, 2008. [Online]. Available: http://www.cs.wisc.edu/condor

[23] 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.

[24] G. B. Dantzig, *Linear Programming and Extensions.* Princeton, NJ: Princeton University Press, 1963.

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

[26] I. David and U. Yechiali, "A time-dependent stopping problem with application to live organ transplants," *Operations Research*, vol. 33, no. 3, pp. 491–504, Jun. 1985.

[27] L. David and I. Puaut, "Static determination of probabilistic execution times," *16th Euromicro Conference on Real-Time Systems (ECRTS '04)*, Jun. 2004.

[28] C. Derman, G. J. Lieberman, and S. M. Ross, "A sequential stochastic assignment problem," *Management Science*, vol. 18, no. 7, pp. 349–355, Mar. 1972.

[29] J. L. Devore, *Probability and Statistics for Engineering and Sciences*, 5th ed. Los Angeles, CA: Duxbury Press, 1999.

[30] S. Dhakal, M. M. Hayat, J. E. Pezoa, C. T. Abdallah, J. D. Birdwell, and J. Chiasson, "Load balancing in the presence of random node failure and recovery,"

The 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), 2006, pp. 25–29.

[31] M. K. Dhodhi, I. Ahmad, and R. Storer, "SHEMUS: Synthesis of heterogeneous multiprocessor systems," *Microprocessors and Microsystems*, vol. 19, no. 6, pp. 311–319, Aug. 1995.

[32] 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.

[33] 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.

[34] M. M. Eshaghian, Ed., *Heterogeneous Computing.* Norwood, MA: Artech House, 1996.

[35] 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.

[36] B. A. Forouzan, *Data Communications and Networking*, 4th ed. New York, NY: McGraw-Hill Science, 2006.

[37] I. Foster and C. Kesselman, Eds., *The Grid 2: Blueprint for a New Computing Infrastructure.* San Francisco, CA: Morgan Kaufmann, 1999.

[38] S. Garg, A. Moorsel, K. Vaidyanathan, and K. S. Trivedi, "A methodology for detection and estimation of software aging," *The Ninth International Symposium on Software Reliability Engineering*, Nov. 1998, pp. 283–292.

[39] C. H. Gebotys and M. I. Elmasry, *Optimal VLSI Architectural Synthesis.* New York, NY: Kluwer Academic Publishers, 1992.

[40] E. Gelenbe, D. Finkel, and S. K. Tripathi, "On the availability of a distributed computer system with failing components," *ACM SIGMETRICS Performance Evaluation Review*, vol. 13, no. 2, pp. 6–13, 2000.

[41] A. Ghafoor and J. Yang, "A distributed heterogeneous supercomputing management system," *IEEE Computer*, vol. 26, no. 6, pp. 78–86, Jun. 1993.

[42] N. Gharachorloo, S. Gupta, R. F. Sproull, and I. E. Sutherland, "A characterization of ten rasterization techniques," *SIGGRAPH '89: Proceedings of the 16th annual conference on computer graphics and interactive techniques.* New York, NY, USA: ACM Press, 1989, pp. 355 368.

[43] S. Ghosh, "Guaranteeing fault tolerance through scheduling in real systems," Ph.D. dissertation, Faculty of Arts and Sciences, University of Pittsburgh, 1996.

[44] C. C. Gonzaga, "Path-following methods for linear programming," *SIAM Review*, vol. 34, no. 2, pp. 167–224, Jun. 1992.

[45] D. Gu, F. Drews, and L. R. Welch, "Robust task allocation for dynamic distributed real-time systems subject to multiple environmental parameters," *25th International Conference on Distributed Computing Systems (ICDCS 2005)*, Jun. 2005, pp. 675–684.

[46] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 3rd ed. San Francisco, CA: Morgan Kaufmann, 2003, ch. 8.

[47] 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.

[48] C.-T. Hwang, J.-H. Lee, and Y. C. Hsu, "A formal approach to the scheduling problem in high level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 4, pp. 464–475, Apr. 1991.

[49] 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.

[50] 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.

[51] H. A. James, K. A. Hawik, and P. D. Coddington, "Scheduling independent tasks on metacomputing systems," 12^{th} *International Conference on Parallel and Distributed Computing Systems (PDCS 99)*, Aug. 1999, pp. 47–52.

[52] F. Junyent, V. Chandrasekar, D. McLaughlin, S. Frasier, E. Insanic, R. Ahmed, N. Bharadwaj, E. Knapp, L. Krnan, and R. Tessier, "Salient features of radar nodes of the first generation NetRad system," *IEEE International Geoscience and Remote Sensing Symposium 2005 (IGARSS '05)*, Jul. 2005, pp. 420–423.

[53] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42–51, Jul. 1998.

[54] W. Keuffer, *Visual Coding with Genitor.* San Francisco, CA: Miller Freeman, 1997.

[55] 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*, vol. 6, no. 3, pp. 281–296, Jul. 2006.

[56] J.-K. Kim, H. J. Siegel, A. A. Maciejewski, and R. Eigenmann, "Dynamic mapping in energy constrained heterogeneous computing systems," *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, Apr. 2005.

[57] 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, "Dynamically mapping tasks with priorities and multiple deadlines in a heterogeneous environment," *Journal of Parallel and Distributed Computing*, vol. 67, no. 2, pp. 154–169, Feb. 2007.

[58] D. Knuth, *The Art of Computer Programming.* Reading, MA: Addison-Wesley, 1997.

[59] A. Kumar and R. Shorey, "Performance analysis and scheduling of stochastic fork-join jobs in a multicomputer system," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 10, Oct. 1993.

[60] C. Leangsuksun, J. Potter, and S. Scott, "Dynamic task mapping algorithms for a distributed heterogeneous computing environment," *4th IEEE Heterogeneous Computing Workshop (HCW '95)*, Apr. 1995, pp. 30–34.

[61] H. M. Lee, S. H. Chin, J. H. Lee, D. W. Lee, K. S. Chung, S. Y. Jung, and H. C. Yu, "A resource manager for optimal resource selection and fault tolerance service in grids," 10^{th} *IEEE International Symposium on Cluster Computing and the Grid*, 2004.

[62] A. Leon-Garcia, *Probability & Random Processes for Electrical Engineering.* Reading, MA: Addison Wesley, 1989.

[63] L. Li, K. Vaidyanathan, and K. Trivedi, "An approach for estimation of software aging in a web server," *The 2002 International Symposium on Empirical Software Engineering (ISESE'02)*, 2002, pp. 91–100.

[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] 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.

[66] P. Manghwani, J. Loyall, P. Sharma, M. Gillen, and J. Ye, "End-to-end quality of service management for distributed real-time embedded applications," 19^{th}

International Parallel and Distributed Processing Symposium (IPDPS 2005), Apr. 2005.

[67] E. Marshall, "Fatal error: How patriot overlooked a scud," *Sience*, pp. 13–19, Mar. 1992.

[68] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementation.* New York, NY: John Wiley & Sons, 1990.

[69] 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.

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

[71] 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.

[72] T. Nakai, "A sequential stochastic assignment problem in a partially observable markov chain," *Mathematics of Operations Research*, vol. 11, no. 2, pp. 230–240, May 1986.

[73] G. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization.* New York, NY: John Wiley & Sons, 1999.

[74] (2007) Opnet technologies, inc. Accessed Feb. 20, 2007. [Online]. Available: http://www.opnet.com/

[75] S. I. Park, V. Raghunathan, and M. B. Srivastava, "Energy efficiency and fairness tradeoffs in multi-resource, multi-tasking embedded systems," *The 2003 International Symposium on Low Power Electronics and Design (ISLPED'03)*, Aug. 2003, pp. 2–13.

[76] C. L. Phillips, J. M. Parr, and E. A. Riskin, *Signals, Systems, and Transforms.* Upper Saddle River, NJ: Pearson Education, 2003.

[77] K. Ramamritham, J. Stankovic, and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Transactions on Computers*, vol. 38, no. 8, pp. 1110–1123, Aug. 1989.

[78] B. Ravindran, R. K. Devarasetty, and B. Shirazi, "Adaptive resource management algorithms for periodic tasks in dynamic real-time distributed systems," *Journal of Parallel and Distributed Computing*, vol. 62, no. 10, pp. 1527–1547, Oct. 2002.

[79] R. Righter, "Stochastically maximizing the number of successes in a sequential assignment problem," *Journal of Applied Probability*, vol. 27, no. 2, pp. 351–364, Jun. 1990.

[80] V. Saario, "Limiting properties of the discounted house-selling problem," *European Journal of Operational Research*, vol. 20, no. 2, pp. 206–210, May 1985.

[81] K. Sakaguchi, "A sequential stochastic assignment problem associated with a non-homogeneous markov process," *Japanese Journal of Mathematics*, vol. 29, pp. 13–22, 1984.

[82] ——, "Best choice problems for randomly arriving offers during a random lifetime," *Japanese Journal of Mathematics*, vol. 31, pp. 107–117, 1986.

[83] Seti at home. Accessed March 31, 2008. [Online]. Available: http://setiathome.berkeley.edu

[84] R. Shcahan, L. Lipsky, and P. Fiorini, "The effect of different failure recovery procedures on the distribution of task completion times," *IEEE Workshop on Dependable Parallel, Distributed and Network - Centric Systems (DPDNS05)*, Apr. 2005.

[85] 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 environment," *14th Heterogeneous Computing Workshop (HCW 2005). In the proceedings of 19th International Parallel and Distributed Processing Symposium (IPDPS 2005)*, Apr. 2005, pp. 122–127.

[86] V. Shestak, J. Smith, A. A. Maciejewski, and H. J. Siegel, "Iterative algorithms for stochastically robust static resource allocation in periodic sensor driven clusters," *8th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2006)*, Nov. 2006, pp. 166–174.

[87] ——, "A stochastic approach to measuring the robustness of resource allocations in distributed systems," *International Conference on Parallel Processing (ICPP 06)*, Aug. 2006, pp. 459–470.

[88] 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 distributed systems," *the 2006 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'06)*, Jun. 2006.

[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*, vol. 68, no. 8, pp. 1157–1173, Aug. 2008.

[90] V. V. Shestak, L. D. Teklits, S. Price, J. Smith, H. J. Siegel, and P. V. Sugavanam, "Methods and systems for improved printing system sheet side dispatch in a clustered printer controller," U.S. Patent 920 060 015, Sep. 1, 2006.

[91] I. Shin and I. Lee, "Periodic resource model for compositional real-time guarantees," 24^{th} *IEEE Real-Time System Symposium (RTSS 2003)*, Dec. 2003, pp. 469–474.

[92] H. Singh and A. Youssef, "Mapping and scheduling heterogeneous task graphs using genetic algorithms," 5^{th} *IEEE Heterogeneous Computing Workshop (HCW '96)*, 1996, pp. 86–97.

[93] J. Smith, L. D. Briceño, A. A. Maciejewski, and H. J. Siegel, "Measuring the robustness of resource allocations in a stochastic dynamic environment," *21st International Parallel and Distributed Processing Symposium (IPDPS 2007)*, Mar. 2007.

[94] J. Smith, V. Shestak, H. J. Siegel, S. Price, L. Teklits, and P. V. Sugavanum, "Resource allocation in a cluster based imaging system," *2007 International Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA 2007)*, Jun. 2007.

[95] Standard performance evaluation corporation. Accessed Feb. 6, 2006. [Online]. Available: http://www.spec.org/

[96] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan, "Distributed job schedulingon computational grids using multiple simultaneous requests," 11^{th} *IEEE International Sumposium on High Performance Distributed Computing (HPDC-11)*, Jul. 2002, pp. 359–368.

[97] 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.

[98] X. Tang and S. T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," *Proceedings of the International Conference on Parallel Processing 2000 (ICPP'00)*, 2000, p. 373.

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

[100] L. Wasserman, *All of Statistics: A Concise Course in Statistical Inference.* New York, NY: Springer Science+Business Media, 2005.

[101] J. P. Watson, L. Barbulescu, and L. D. Whitley, "Contrasting structured and random permutation flow-shop scheduling problems: Search-space topology and algorithm performance," *INFORMS Journal on Computing*, vol. 14, no. 2, pp. 98–123, Apr. 2002.

[102] D. Whitley, "The genitor algorithm and selective pressure: Why rank-based allocation of reproductive trials is best," *3rd International Conference on Genetic Algorithms*, Jun. 1989, pp. 116–121.

[103] L. A. Wolsey, *Integer Programming.* New York, NY: John Wiley & Sons, 1998.

[104] A. Wood, "Predicting client/server availability," *Computer*, vol. 28, no. 4, pp. 41–48, Apr. 1995.

[105] M. Wu and W. Shu, "Segmented min-min: A static mapping algorithm for meta-tasks on heterogeneous computing systems," *Proceedings of the 9th IEEE Heterogeneous Computing Workshop*, Mar. 2000, pp. 375 385.

[106] 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.

[107] T. Yamada and C. Reeves, "Permutation flowshop scheduling by genetic local search," *Genetic Algorithms in Engineering Systems: Innovations and Applications*, Sep. 1997, pp. 232–238.

[108] T. Yuan, J. Boangoat, E. Ekici, and F. Ozguner, "Real-time task mapping and scheduling for collaborative in-network processing in dvs-enabled wireless sensor networks," 20^{st} *International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Apr. 2006, pp. 21–27.