Resource Allocation and Scheduling Strategies using Utility and the Knapsack Problem on Computational Grids

by

Daniel Colin Vanderster
B.Eng., University of Victoria, 2003

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Electrical and Computer Engineering

University of Victoria

Resource Allocation and Scheduling Strategies using Utility and the Knapsack Problem on Computational Grids

by

Daniel Colin Vanderster
B.Eng., University of Victoria, 2003

Supervisory Committee

Dr. Nikitas J. Dimopoulos, Supervisor
Department of Electrical and Computer Engineering

Dr. Randall J. Sobie, Supervisor and Outside Member
Department of Physics and Astronomy

Dr. Kin F. Li, Departmental Member
Department of Electrical and Computer Engineering

Dr. Amirali Baniasadi, Departmental Member
Department of Electrical and Computer Engineering

Dr. Rajkumar Buyya, External Member
Department of Computer Science and Software Engineering,
University of Melbourne

Supervisory Committee

Dr. Nikitas J. Dimopoulos, Supervisor
Department of Electrical and Computer Engineering

Dr. Randall J. Sobie, Supervisor and Outside Member
Department of Physics and Astronomy

Dr. Kin F. Li, Departmental Member
Department of Electrical and Computer Engineering

Dr. Amirali Baniasadi, Departmental Member
Department of Electrical and Computer Engineering

Dr. Rajkumar Buyya, External Member
Department of Computer Science and Software Engineering,
University of Melbourne

Abstract

Computational grids are distributed systems composed of heterogeneous computing resources which are distributed geographically and administratively. These highly scalable systems are designed to meet the large computational demands of many users from scientific and business orientations. This dissertation address problems related to the allocation of the computing resources which compose a grid.

First, the design of a pan-Canadian grid is presented. The design exploits the maturing stability of grid deployment toolkits, and introduces novel services for efficiently allocating the grids resources. The challenges faced by this grid deployment motivate further exploration in optimizing grid resource allocations.

The primary contribution of this dissertation is one such technique for allocating grid resources. By applying a utility model to the grid allocation options, it is possible to quantify the relative merits of the various possible scheduling decisions. Indeed, a number of utility heuristics are derived to provide quality-of-service policies on the grid; these implement scheduling policies which favour efficiency and also allow users to intervene with urgent tasks. Using this model, the allocation problem is then formulated as a knapsack problem. Formulation in this manner allows for rapid solution times and results in nearly optimal allocations.

The combined utility/knapsack approach to grid resource allocation is first presented in the allocation of single resource type, processors. By evaluating the approach with novel utility heuristics using both random and real workloads, it is shown to result in efficient schedules which have characteristics that match the intended policies. Additionally, two design and analysis techniques are performed to optimize the design of the utility/knapsack scheduler; these techniques play a significant role in practical adoption of the approach.

Next, the utility/knapsack approach is extended to the allocation of multiple resource types. This extension generalizes the grid allocation solution a wider variety of resources, including processors, disk storage, and network bandwidth. The general technique, when combined with new heuristics for the varied resource types, is shown to result in improved performance against reference strategies.

This dissertation concludes with a novel application of the utility/knapsack approach to fault-tolerant task scheduling. Computational grids typically feature many techniques for providing fault tolerance to the grid tasks, including retrying failed tasks or replicating running tasks. By applying the utility/knapsack approach, the relative merits of these varied techniques can be quantified, and the overall number of failures can be decreased subject to resource cost considerations.

Table of Contents

List of Tables

List of Figures

Acknowledgements

My time as a graduate student at UVic has been supported by many kind people. This page, undoubtedly incomplete, attempts to recognize those who have been most helpful along the way.

First of all, I want to thank my graduate supervisors, Randy Sobie and Nikitas Dimopoulos. Their guidance, support, and encouragement have helped me immensely throughout the past five years.

My research has been helped along by many collaborators. I thank Ashok Agarwal, who initially introduced me to the toolboxes of grid computing, and later introduced me to the cultures and cuisines of India. Next, I gratefully acknowledge Rafael Parra-Hernandez, whose work provided the spark that set my early research alight. My other friends in the LAPIS lab, especially Nainesh Agarwal and Farshad Khunjush, were quite helpful with my duties as a teaching assistant. Finally, I appreciate the help of my Grid Canada and GridX1 collaborators, notably Andre Charbonneau, Ron Desmarais, Roger Impey, Gabriel Mateescu, Wayne Podaima, Darcy Quesnel, and Rod Walker.

Outside the lab, I am greatly indebted to my close friends Ryan Enge and Ian Gable, with whom I have shared many adventures. My best man, Ryan, worked with me on every single lab and assignment throughout my undergraduate years (except one or two, maybe). And Ian was with me in the beginning (living two doors down, waking me up to go to calculus), in the middle (when he encouraged Randy to hire me), and is still around today. I also thank Aaron Sanderson for many enlightening conversations about rockets, reality TV, and taxes.

Most of all, I thank my family. I am infinitely grateful to my parents, Sissel

and Ron, for giving me everything while wanting nothing. They encouraged me to continue my studies as long as I could, and gave me the confidence to take on all challenges.

Finally, I thank Laura, who deserves more acknowledgement than can fit on this page. Her unquestioning support and uncanny wisdom has been invaluable though the ups and downs of this journey. And lastly, I thank her for Judah.

For my parents, Ron and Sissel,

and my love, Laura

Chapter 1

Introduction

The field of grid computing encompasses a variety of concepts, all of which are related to the scalable sharing of heterogeneous computing resources across large geographical distances and administrative domains. Computational grids are one specific type of grid; their primary purpose is to present a single, transparent, interface to the processing resources at many facilities. The key challenges in developing these systems are both practical and academic in nature; this dissertation addresses problems in both of these areas. In the first part, the design of a pan-Canadian computational grid is presented, focusing on an architecture which integrates both preexisting and new components. The presented design is shown to be both useful and reliable to scientific users, and it motivates further investigations into the optimization of grid resource allocations.

The second part of this dissertation introduces a new resource allocation strategy which provides a means of implementing quality of service (QoS) policies on the grid. The strategy is based around the concept of options; when a task is submitted to the grid, there are many options available to allocate resources to the task. By applying a utility model to these options, it is possible to quantify the relative merits of various scheduling decisions available for a given task. The allocation problem is then formulated as a knapsack problem; this allows for quick and accurate solutions to

finding nearly optimal allocations. When this utility/knapsack approach is applied to varied grids and resource types, it is shown to be effective in improving performance and reliability while allowing users to intervene with urgent tasks.

This dissertation are organized into seven chapters as follows.

Chapter 2: Background and Related Research

Chapter 2 provides an overview of grid computing, its components and standards, and surveys the field of resource allocation and task scheduling. After first describing the main components of a computational grid, the key technologies and standards are described. The second part of chapter 2 defines the problem of grid resource allocation and surveys a variety of approaches to the solution of this problem. Finally, two of the main techniques used in this dissertation are reviewed: utility models and the knapsack problem.

Chapter 3: GridX1, A Pan-Canadian Computational Grid

Chapter 3 describes the design and usage of GridX1, the first computational grid deployment project in Canada. This project, although being motivated by the needs of the Canadian particle physics community, presents a general purpose grid design which can fulfil the computational needs of many scientific fields. The design of GridX1 demonstrates a deployment of the *de facto* standard Globus Toolkit and introduces a novel metascheduler service which allows for centralized access to all of the computational resources. Examples of the system's usage are drawn from the ATLAS and BaBar particle physics applications.

Chapter 4: Grid Resource Allocation using a Utility Model and Knapsack Formulation

In chapter 4, a new solution to the problem of allocating a grid's processing resources is presented. Using a model which evaluates the utility, or merit, of each allocation option for a given task, the allocation problem is formulated as a variant of the

0-1 multichoice multidimensional knapsack problem. A number of utility heuristics are introduced and evaluated using a variety of task workloads. The resulting task schedules are shown to be consistent with each heuristic's intentions, thereby validating the knapsack approach, and demonstrate improved performance and efficiency in comparison with reference strategies.

Chapter 5: Design and Analysis of the Utility/Knapsack Scheduling System

Chapter 5 presents two methodologies which can be used to optimize the design of the utility/knapsack approach. First, sensitivity analyses are performed to determine the effects of each utility heuristic used at varied weights in a utility function. Next, a more detailed study uses the Plackett-Burman design methodology to find the main effects of each parameter in a given utility function. Each of these design techniques delivers key insights into the utility/knapsack approach, and are thus shown to be critical to the practical application of this system.

Chapter 6: Allocation of Multiple Resource Types

In chapter 6, the allocation problem is extended to resources of many types. In particular, the utility/knapsack approach is used to allocate both processing and storage resources, and introduces utility heuristics which incorporate aspects of both resources. The allocation problem is formulated as 0-1 multichoice multidimensional knapsack problem in order to optimize the overall utility subject to multiple resource constraints. The utility/knapsack approach is demonstrated to perform well in the multidimensional case, showing improvements in overall efficiency and quality of the resulting task schedules.

Chapter 7: Selection between Fault Tolerance Options using the Utility/Knapsack Approach

Chapter 7 addresses the problem of selecting between fault tolerance techniques available to an application. The work investigates a system which selects between two such techniques: retrying failed tasks and replicating running tasks. Noting that each technique presents varied costs to the resource owners, the utility/knapsack approach is applied using three utility heuristics. Each heuristic is shown to result in allocations which both decrease task failures and improve the overall performance of the grid.

Chapter 8: Conclusions

Chapter 8 summarizes the key results of the dissertation and presents directions for future research.

Chapter 2

Background and Related Research

This chapter serves three purposes. First, it provides unfamiliar readers with necessary background of the field of grid computing, including descriptions of the key concepts and components. Second, this chapter introduces the problem of allocating tasks on computational grids, and discusses the key strategies and heuristics which are available in the literature. Finally, it provides an introduction to utility models and the knapsack problem, which are two techniques employed in this dissertation.

2.1 Introduction to Grid Computing

Grid computing encompasses a large variety of concepts related to the sharing of computational resources. The Grid [1], in the proper sense of the term, is an idealized system which provides users with transparent access to vast resources of many types, including processors, storage, networks, instruments, and actuators (see figure 2.1). The usage of the term "grid" to describe this system was drawn from the power grid, a system which is analogous in that it provides transparent access to electrical power on demand. In both grids, the source of the power is unknown to the user; as long as their demands are met, they are not concerned with the details of the system.

Grid computing in the present involves the scalable and transparent sharing of heterogeneous computing resources across large geographical distances and administrative boundaries. The central themes of grid computing are detailed as follows.

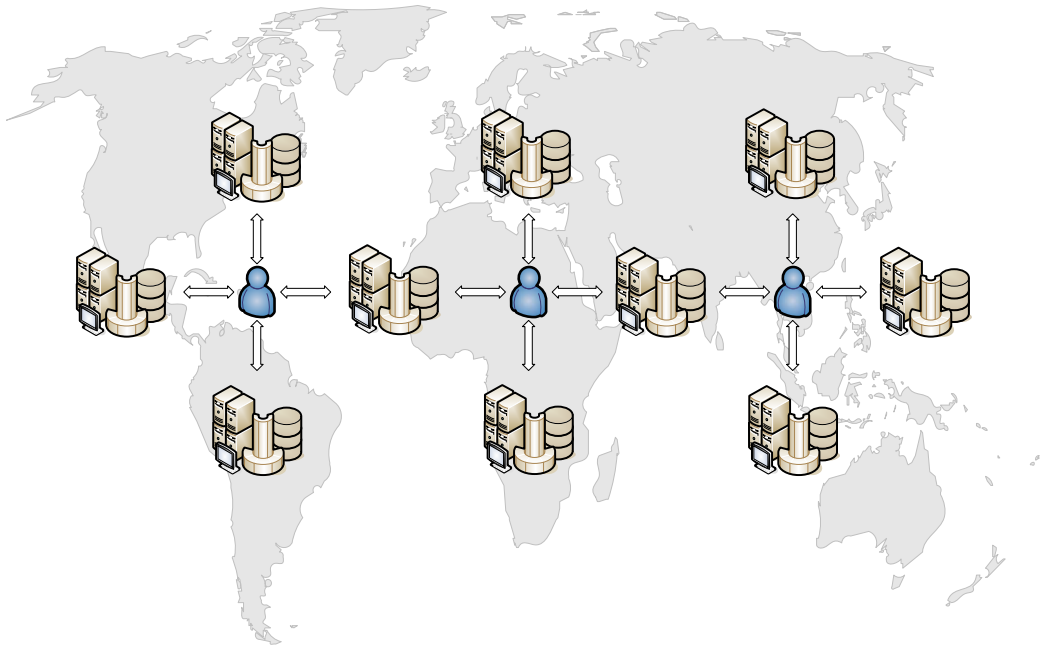

Figure 2.1: The Grid provides users with access to distributed resources of many types.

Sharing There are many problems in scientific study that require more computational capabilities than those which are available at a single facility. Consider, for example, the ATLAS project at the Large Hadron Collider at CERN in Geneva, Switzerland. Expected to turn on in 2008, the particle detector will generate many petabytes of data per year [2]. As a result of issues related to reliability, data protection, and the distribution of funding, the system to store and process this data must be distributed across facilities around the world. Because a given block of data could be located at any of these facilities at a given time, access to the facilities must be shared among the users of the system. The sharing of these resources occurs across large geographical distances, implying large network latencies which make interactivity difficult to achieve. In addition, the sharing of resources occurs across administrative boundaries, which results in a wide variety of problems related to security. In general, the concept of sharing in grid computing delivers many of the central challenges in the field.

Scalability Grids need to scale in two ways. First, they need to be able to scale up in capability in order to meet the escalating demands of computationally-bound applications (e.g. the analysis of particle collisions). Second, they must be able to manage large numbers of users in such a way that unique and specific requirements of each user can still be met (e.g. providing world-wide access to a rare instrument). These scalability requirements result in a reliance on innovative methods for improving the efficiency and reliability of the grid.

Heterogeneity Grids are composed of many different types of resources, each of which is available over a range of capabilities. Making a variety of resource types available to grid users is one of goals of the idealized Grid. Consider that a scientific computing job is often described as a workflow; a workflow is a directed acyclic graph where each node corresponds to a sub-task. At each step in the workflow, the job makes use of varied resources on the grid, whether it is sensing an instrument, moving data, processing raw information, or creating visualizations. Having a range of resources in a grid makes complex and efficient workflows possible.

At a lower level, the resources of a given type are also heterogeneous, with different components of the grid providing a varied capability or performance. For the most prevalent of resources, such as storage and processors, the rapid growth in technological performance ensures that faster resources are always appearing. The variety of capabilities throughout the grid provides many opportunities for optimizing overall performance; for example, in a simple system, high priority tasks can be directed to the fastest processors, or an economically-driven grid could provide discounted access to the slowest.

Transparency The final feature of grids, being motivated by usability and performance, is transparency. The idealized Grid provides access to a wide variety of users through a single, easy-to-use, interface. Because end-users interact with this single interface, the grid must provide a mechanism for acting on behalf of the user at many

sites. This feature, known as single-sign-on, requires a security infrastructure which provides for time-limited authentication, scalable authorization frameworks, and a permissions delegation mechanism.

2.2 Computational Grids: Components, Standards, and Toolkits

Ideally, the Grid is a large system which meets the widely varied demands of many different users; however, such a system is not yet feasible in practise. Instead, many more focused grids have developed to meet specific needs. One specific type is the *data grid* [3], whose primary focus is on storing and managing information in a manner which is secure, reliable, and fast. Another type of grid is the *collaborative grid* [4], which provides visualization tools to make remote collaboration possible.

This dissertation is directed at a third type of grid, one whose primary purpose is to provide computational or processing capacity: the *computational grid*. The capabilities provided by a computational grid could be considered as a subset of those of the idealized Grid, implying that any improvements that can be made to computational grids are also applicable to the ideal Grid as well.

The basic unit in a computational grid is the processor, which is typically housed in a single or multi-processor *worker node*. These nodes are commonly joined at the institutional level to form *clusters*, often running a variant of the Unix or Linux operating systems. These clusters are then joined at the national or international level using *middleware*, a software suite of protocols and tools, to form a computational grid. Computational grids are alternatively referred to as compute grids, multi-clusters, or utility grids, which are a commercial variant.

2.2.1 Components of a Computational Grid

Computational grids are composed of a number of services which can be classified into four categories: job management, data management, information systems, and

security.

Job Management: The job management services are the most visible services provided by a computational grid. There are two basic services in this category: the grid interface and the cluster interface. The grid interface is a central or distributed service which is accessed by end-users to submit, monitor, cancel, or otherwise manage their jobs. One component of the grid interface is the *resource broker*, also known as a *metascheduler*. This component is exclusively responsible for the management of the grid resources, including the matching of tasks to resources. The cluster interface is used by the grid to relay jobs between the grid interface and the clusters. This component often supports the same features as the grid interface, including submitting, monitoring, and cancelling jobs.

Data Management: A secondary group of services is responsible for managing the data used by the computational grid. These data management services include mechanisms for the secure, reliable, and fast transmission of data, as well indexing data locations and metadata in catalogues. Techniques such as replication, where multiple copies of the same data are stored at various sites in the grid, are often used to improve performance and reliability.

Information Systems: The efficient utilization of the grid relies on accurate and up-to-date information about the status of its resources. A variety of information services provide this functionality. Typically, a resource status service is used to monitor the availability, functionality, and capacity of the grid resources; this information can be obtained using a push or pull model, and probing technologies such as agents can also be used. In addition, the status of the grid services themselves are monitored; this is required for accounting, auditing, and functionality testing purposes. The up-to-date status of the grid

is often made available to users and operators via a web-based graphical user interface.

Security: The final and perhaps most critical supporting services are those that provide for security in a computational grid. Secure protocols are necessary to provide the most basic service on the computational grid, the single-sign-on interface. This interface requires a mechanism for authentication at the interface as well as delegation of the authenticated credentials to other grid sites. The authentication mechanism requires a vouching authority which verifies the credibility of each user's identity. Each grid resource then maintains a mapping of authenticated users to their permissions, or authorizations, at the resource. Geographically and administratively distinct users often belong to one or more groups, each of which is characterized by the common goals, requirements, and hence permissions of its constituents; these are referred to as a Virtual Organization (VO) [5] of users. In this case, grid resources map VO's to local privileges; this saves every resource from the unwieldy task of having to maintain a database of permissions for each individual.

2.2.2 Grid Standards and Toolkits

In general, the grid computing community is focused on the development of open standards. Early in the development of grid computing, the Globus Alliance sought to develop a complete grid architecture. This led to the development of a suite of tools and protocols for grid development known as the Globus Toolkit [6]. Due to a combination of its quality, openness, and an absence of viable competing solutions, the protocols and tools developed by Globus became *de facto* standards. The Globus Toolkit demonstrates that a small number of protocols and tools can be leveraged to make diverse resources available to many applications. The key protocols developed by Globus and implemented in version 2 of their Toolkit (GT2) include:

GSI: The Grid Security Infrastructure [7] uses public key cryptography to provide security to the grid. Each user and service is authenticated using a certificate. The certificates are encoded in the X.509 format [8] and are signed by a certificate authority which verifies the authenticity of the certificate. The GSI provides single-sign-on and credential delegation through its notion of proxy certificates, which once generated using a secret password are valid for a limited time.

GASS and GridFTP: Global Access to Shared Storage [9] and the Grid File Transfer Protocol [10] are two data services developed by Globus to provide secure and rapid transmission of data between resources. Both protocols use the GSI for authentication, however GridFTP provides advanced features such as striping of data across many network connections to improve performance and third party transfers of data between two sites but initiated from a third site.

GRAM: The Grid Resource Allocation and Management [11] service is responsible for submitting and managing jobs on the local resource management system of a grid cluster. It works together with the GSI to delegate credentials to the targeted clusters and with the data services for staging files from the submission computer to the execution host.

MDS, GRIS, and GIIS: The Monitoring and Discovery System [12] is an information service built around the Lightweight Directory Access Protocol [13]. Using a Grid Resource Information Service, users and applications can directly query the resource's status as it is reported by a local information provider. In addition, resources can publish their availability to an aggregate directory called the Grid Index Information Service. Using the GIIS, users and applications can discover the availability and status of all of the grid resources.

GT2 is a very successful middleware, allowing for the development of production quality grids around the world. However, the developing nature of the field brought about new challenges that could not be solved using GT2's rather inflexible protocols. These limitations led to collaboration with the Web Services community.

Web Services (WS) is a modular and extensible system based on open standards for making arbitrary services available on networks which use the Simple Object Access Protocol (SOAP) over the HyperText Transfer Protocol (HTTP) [14]. One of the driving principles behind WS is statelessness; a given service has no internal memory, so its output is a function of only the arguments used in its invocation. Grid services, however, require an internal memory to keep track of the resources a service represent. A stateful extension to WS, the Web Services Resource Framework (WSRF) [15], was therefore developed by the Globus Alliance in collaboration with IBM and HP. By including WS versions of the GT2 protocols, WSRF provides the basic framework for building a grid. However, by building on the WS standards, new functionality can be more easily developed. Version 1.2 of the WSRF was approved as by the OASIS organization in April 2006 as an OASIS standard [16].

2.3 Resource Allocation and Task Scheduling on Computational Grids

One of the most critical components of the interface to a computational grid is the resource broker. This service, which is alternatively referred to as a metascheduler due to its function as a scheduler of schedulers, is responsible for the allocation of the grid resources to the resource requesters, which are typically in the form of *jobs* or *tasks*. Each task specifies its resource demand, typically the number of processors requested, the memory required, the execution environment required, and others. Tasks may or may not be malleable, meaning that they can vary the number of processors used (e.g. prefer 64 processors, but run slower on 32). These tasks are submitted by

an end-user to the grid interface which uses the resource brokering component to place the task on the grid. The optimization of this placement according to various measures is the primary subject of this dissertation.

The process of selecting a set of tasks to available resources relies on the availability of information about the task and the resources. Some of the task information which can be used in scheduling decisions include:

- requirements,
- submission time,
- priority,
- estimated running time, and
- owner.

In addition, information about the resources that can be used by the scheduler include:

- availability,
- capacity,
- queue length,
- estimated wait time,
- architecture,
- performance,
- operating system,
- software and libraries, and
- time of day.

The literature describes a number of techniques for resource allocation on the grid. Many of these pre-date grid computing; they are results of research in operations and early parallel computers. Notable techniques are described here.

2.3.1 First-come-first-served

The most simple scheduling strategy is first-come-first-served (FCFS). In this strategy, tasks are assigned in the order they arrive in a queue to an arbitrary resource which meets the task's requirements. The FCFS strategy implies that the priority of a task is equal to the time it has spent waiting in the queue (current time minus submission time).

2.3.2 Backfilling

Backfilling is an effective strategy for scheduling parallel tasks efficiently. In systems of parallel tasks, the job at the head of the queue is often delayed because of a lack of available resources. However, there are often less demanding jobs in the queue which could be executed earlier without delaying the job at the head. EASY backfilling follows this strategy [17]. If the resources for the job at the head of the queue are expected to become available at time t, then the EASY backfilling strategy allows a job with expected length l, where $l < t$, to be executed immediately, rather than waiting, assuming its required resources are available. This effectively fills the idle bubbles in the schedule and results in a shorter overall completion time.

2.3.3 Makespan Minimizing Heuristics

The makespan of a set of tasks is the time taken between the start of the first task and the completion of the last task. Finding the minimal makespan is nondeterministic polynomial-time hard (NP-hard) [18]. As such, research in operations and heterogeneous computing has resulted in a number of heuristic approaches to minimizing the makespan [19]. Examples include the following:

Min-Min The min-min heuristic assigns the shortest task to the resource that will execute it the fastest [18, 19]. The motivation behind min-min is that it seeks to cause the smallest change possible to the resources during the assignment of each task.

Max-Min The max-min heuristic is similar to min-min, but instead prefers to run the longest tasks early. In the situation where there are many short tasks and few long tasks, then the min-min heuristic results in low efficiency and suboptimal overall completion time. Because min-min postpones the long tasks, they are left running on a small proportion of the processors, leaving the remaining processors to idle. The overall completion time is therefore dramatically increased for such sets of tasks.

Other Heuristics Many other heuristics exist for the minimization of the makespan [19]. Some of these include the suffrage techniques [20, 21], genetic algorithms, synthetic annealing, and others.

2.3.4 Gang Scheduling

Gang scheduling [22] is a strategy which is used to improve interactivity in shared parallel computer systems by using time slicing. Time slicing is a technique commonly used in personal computers to provide the illusion of multitasking on a single processor system. Because each task is limited to its given time slice, gang scheduling has the added benefit of limiting the negative effects of bad scheduling decisions.

2.3.5 Matchmaking

Matchmaking [23] is the resource allocation framework employed by Condor [24] and Condor-G [25]. In this strategy, each consumer (job) and provider (resource) provides a classified advertisement (ClassAd), which contains the entity's description and characteristics. In each ClassAd, the entities provide Requirements and Rank expressions: Requirements is a boolean expression used to specify constraints (e.g. that a job has for potential resources, or vice versa); and Rank is real number used to describe the preferences (e.g. of a resource for jobs with certain characteristics, or vice versa.) The Rank and Requirements are functions of the metadata contained in the job and resource ClassAds. During the matchmaking algorithm, the Rank and Requirements expressions are evaluated for each task-resource combination; the

match which has a satisfied Requirements expression (Requirements==TRUE), and having the largest Rank is selected.

Matchmaking is a practical solution to the problem of specifying allocation preferences. However, matchmaking itself does not result in any resource allocation optimizations; rather, its value is that it provides a framework within which such optimization heuristics can be implemented. Further, the Rank mechanism is somewhat inflexible, as it does not easily allow for access to the state of the scheduler itself (e.g. the implementation of a round-robin placement algorithm is non-trivial.)

2.3.6 The Economic Grid and Market-based Resource Allocations

An alternative and notable variation on the computational grid is the economic grid. The market-based allocation strategies of Buyya [26] were first developed for the Nimrod-G resource broker [27] and are now available in the tools provided by the Gridbus project [28]. In this system, the grid is implemented with economical considerations: resources have associated costs, and users pay to use them. Buyya shows how to apply various market-based pricing models to the grid, including flat pricing, auction style pricing, timing models (peak/off-peak), supply/demand models, and others. The Nimrod-G resource broker uses scheduling algorithms which minimize the total cost subject to timing constraints, or minimize the total time subject to budget constraints. The cost optimizing algorithm sorts the available resources by their cost, and assigns tasks to the cheapest resources first while ensuring their deadline is not exceeded. The time optimizing algorithm sorts the resources by the estimated time until they become available, and assigns tasks to the resource with the soonest availability while ensuring their budget is not exceeded.

In addition, the Libra scheduling system for clusters [29, 30] incorporates a concept of utility into the scheduling decisions. In this system, the utility of a task is related to the budget allocated to the job, the deadline of the task, and any service-level-agreements (SLA's) the task might have with resources. Libra schedules by assigning

each task to a fraction of a processor in such a way that the task is likely to meet its budget and deadline constraints.

Though these techniques are not directly applicable to the non-economic grid, they represent a significant contribution to computational grid research.

2.4 Utility Models and the Knapsack Problem

The central contributions of this dissertation involve the introduction and analysis of grid resource allocation using a *utility model* to express allocation preferences and the formulation of the allocation problem as a *knapsack problem*. Whereas both utility models and knapsack resource allocation formulations are readily available in the literature [31, 32], this work represents a new and significant solution to the problem of grid resource allocation using a combination of both techniques.

2.4.1 The Utility Model

The concept of *utility* is commonly used in allocation, admission, and other problems related to the provision of service guarantees or Quality of Service (QoS). Central to this model is the concept that every *decision* to be made has multiple *options*; decisions in allocation problems result from selection of resources to allocate for a given resource request. Each of these options has a perceived utility (a.k.a. value, merit, usefulness); utility can express the desires of the end-user, the resource owners, or others. In general, the utility u_{ij} of option j of decision i is computed as a function of the metrics $\mathbf{m}_{ij}$ using the utility function U:

$$u_{ij} = U(\mathbf{m}_{ij}) \tag{2.1}$$

A variety of techniques exist for allocating resources using the utility model. For example, decisions could be made in FCFS order using the option which has the largest utility. The following section describes a more accurate solution to the allocation

problem.

2.4.2 The Knapsack Problem

The knapsack problem derives its name from the thief's problem of choosing which of a variety of treasures to steal in such a way that the value of his spoils is maximized and that he doesn't exceed the weight carrying capacity of his knapsack. This problem is referred to as a 0-1 knapsack problem, as opposed to a fractional knapsack problem, because each treasure is indivisible. If the treasures are somehow grouped in such a way that at most one of each group can be selected, it is a multichoice knapsack problem. If the thief has other constraints, such as volume, length, or anything else, it is a multidimensional knapsack problem. The combination of all of these constraints creates a 0-1 multichoice multidimensional knapsack problem (MMKP).

Consider the problem of finding the optimal selection from among n items. If x_{ij} is a variable indicating the selection of option j of item i, then

$$x_{ij} \in \{0, 1\} \tag{2.2}$$

Let l_i be the number of options for item i and let v_{ij} be the value of option j of item i. Further, knapsack k, of m in total, has capacity b_k. The objective is to maximize the sum of the values of the selected options

$$\sum_{i=1}^{n} \sum_{j=1}^{l_i} v_{ij} x_{ij} \tag{2.3}$$

subject to the multidimensional knapsack constraint

$$\sum_{i=1}^{n} \sum_{j=1}^{l_i} a_{ijk} x_{ij} \leq b_k \qquad k = 1, ..., m \tag{2.4}$$

the multichoice constraint

$$\sum_{j=1}^{l_i} x_{ij} \leq 1 \qquad i = 1, ..., n \tag{2.5}$$

and the selection variable constraint in equation 2.2. The solution of the MMKP is NP-hard [33], therefore we rely on heuristics [34] to find solutions in a reasonable amount of time.

Chapter 3

GridX1: A Pan-Canadian Computational Grid

This chapter presents the design and deployment results of GridX1, a pan-Canadian computational grid. GridX1 is designed to unify many of the medium and large scale computational cluster facilities in Canada into a single general-purpose system. It is built around a combination of off-the-shelf grid middleware and introduces a new metascheduler service built around existing tools. After describing the design of GridX1, the deployment results from two particle physics projects, ATLAS and BaBar, are shown to highlight the usefulness of the GridX1 system. Finally, this chapter motivates further research into resource brokering and metascheduling optimizations, which are the topic of the remaining chapters in this dissertation.

The work presented in this chapter is published in the Future Generation Computer Systems journal [35] and represents the combined work of the GridX1 collaborators. My work in this project included contributions in the overall design (section 3.3), metascheduler service (section 3.3.1), monitoring (section 3.3.2), and LCG interface (section 3.4.1)

3.1 Introduction

Computational grids [1] are emerging as the preeminent model for large scale scientific computing. By providing single-sign-on access to heterogeneous resources distributed across administrative domains, grids provide simplified access to vast computing resources. An increasing number of groups around the world are taking part in grid development and deployment, and as a result the efficiency and reliability of grid software is maturing quickly. Notable production grid projects include TeraGrid [36], the Open Science Grid [37], the LHC Compute Grid [38], NorduGrid [39], INFN Grid [40], the UK e-Science Grid [41], and GridLab [42]. These projects have participated in developing comprehensive grid middlewares, including the Virtual Data Toolkit (VDT) [43], the Enabling Grids for E-sciencE (EGEE) project [44] and the Advanced Resource Connector (ARC) project [45].

GridX1 is a collaborative project that has established a computational grid infrastructure across Canada. Canada has a number of medium to large scale computing facilities with a broad range of clusters, parallel, and vector machines. GridX1 unifies some of the cluster resources into a large virtual facility. The resources are not dedicated to GridX1 but instead treat the grid users as another local user. This allows GridX1 to exploit unused cycles at these facilities. The facilities available to GridX1 are shared with many fields of research. Each facility operates independently with its own management team and user community. It is therefore desirable to use grid software which minimizes the management requirements without compromising security or interfering with the normal operation.

The GridX1 infrastructure is built using a combination of off-the-shelf middleware to provide the basic grid services and a novel application of Globus [6] and Condor-G [25] to build a metascheduler. Various strategies for ranking resources are also presented, including an Estimated-Waiting-Time algorithm, a throttled load balancing

strategy, and a novel external ranking strategy based on data location. The design of GridX1 has been focused around the execution of parameter-study applications. The first users of GridX1 have been particle physicists working on the ATLAS experiment for the Large Hadron Collider (LHC) project at CERN and the BaBar experiment at the Stanford Linear Accelerator Center (SLAC).

The LHC Computing Grid (LCG) [38] is being developed to analyze the data that will be collected by the LHC experiments starting in 2008. To meet the Canadian computing commitment for the ATLAS experiment, the easiest solution would be to have Canadian facilities join the LCG directly. While this would be simple and effective, sites wishing to join the LCG are faced with strict resource and policy requirements, including a requirement for dedicated hardware resources. Resource dedication is difficult because many of the Canadian facilities are shared by researchers from many fields. By developing an interface which federates the GridX1 resources into the LCG, we have contributed substantial computing resources to the project while maintaining the shared nature of the facilities.

The primary purpose of this chapter is to provide a complete description of the GridX1 project, highlighting the techniques, tools, and algorithms developed to ensure the scalability, efficiency and reliability of the grid. The remainder of this chapter is organized as follows. An overview of the GridX1 resources is provided in section 3.2. The design and implementation of the GridX1 infrastructure and the introduction of GridX1's novel metascheduling strategies are presented in section 3.3. Results on the performance of GridX1 resulting from the execution of particle physics event simulations are presented in section 3.4. Finally, we conclude in section 3.5.

3.2 GridX1 Resources

GridX1 resources include computational clusters at the Centre for Subatomic Research at the University of Alberta, the Research Computing Centre at the Univer-

sity of Victoria, the WestGrid Facility at the University of British Columbia, the Research Computing Support Group at the National Research Council in Ottawa, McGill University in Montreal, and the University of Toronto. In some instances, sites may have more than one cluster. The total number of processors at these sites is approximately 2500 with disk and tape storage well in excess of 100 TB. GridX1 is given access to a fraction of these resources, with some sites allocating a specific job quota, and others allowing GridX1 to backfill during periods of low utilization.

Each cluster uses 32-bit x86 processors from Intel and AMD ranging in clock speed from 1 to 3.2 GHz. A variety of operating systems are implemented, including RedHat Linux version 7.3 and Enterprise 3, Scientific Linux (RHEL3-compatible), and SUSE Linux. Most sites have deployed the Portable Batch Scheduler (PBS) [46, 47] software for local resource management, with a few deploying the Condor batch system [24]. The sites are configured with one or more head nodes which act as a user interface to the worker nodes. The worker nodes are required to have external network access via network address translation or a similar technique. Most of the sites have 1 gigabit/s network connectivity to the Canadian research network provided by CANARIE.

3.3 The GridX1 Infrastructure

The GridX1 infrastructure builds upon existing mature middleware tools to create a stable and easily maintainable grid. Figure 3.1 depicts the overall infrastructure of GridX1, incorporating the Globus Toolkit version 2 (GT2) [6], MyProxy credential repository [48], and Condor-G resource management system [25]. Although they are readily available individually from their respective organizations, GridX1 has been deployed using these components as packaged in the Virtual Data Toolkit [43]. This middleware package is well-maintained, easy to install, and provides a stable deployment platform. In addition to these basic components, a number of GridX1-

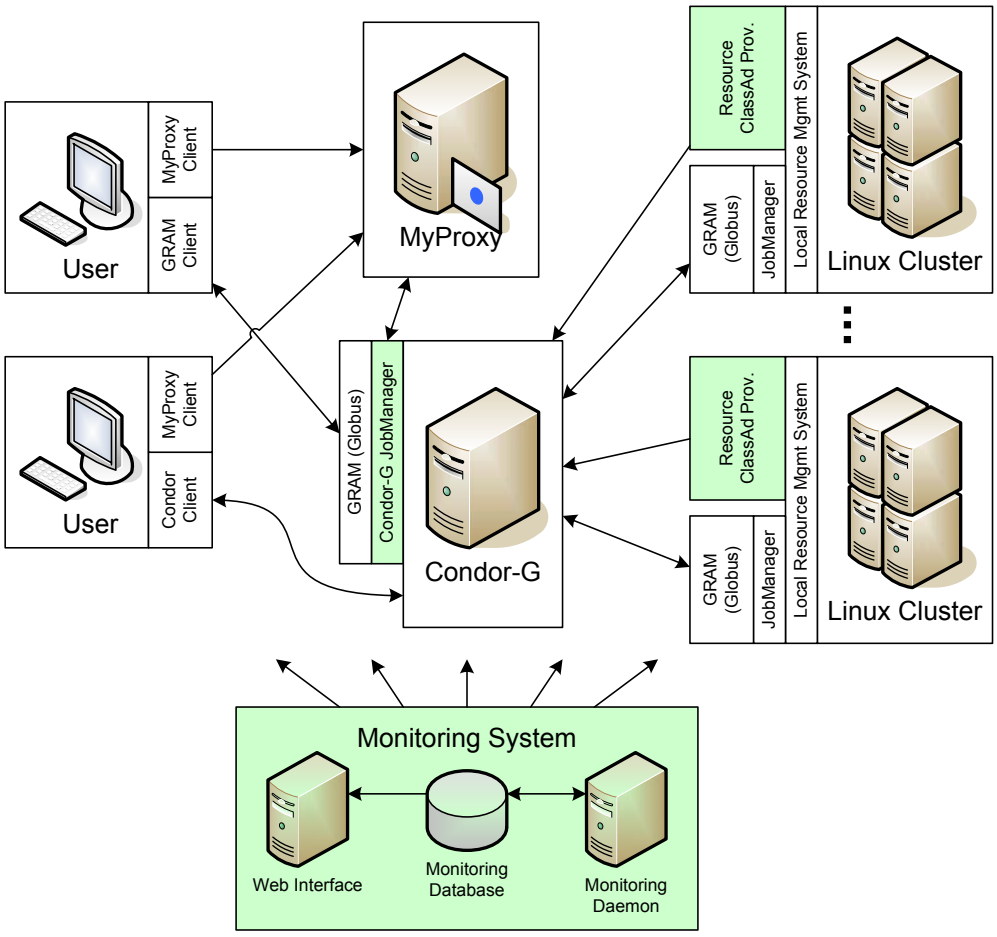

Figure 3.1: The GridX1 infrastructure builds upon off-the-shelf middleware components such as Globus [6], Condor-G [25], and MyProxy [48] and integrates new components (highlighted in green) developed for GridX1.

specific tools have been developed to enhance job distribution and to monitor jobs and resources through a web interface. GridX1 employs the standard Grid Security Infrastructure [7], implemented in the Globus Toolkit, to provide single-sign-on authentication, authorization mechanisms, secure communication, and the auditing of user actions. All GridX1 hosts and users are assigned an X.509 certificate [8] by the Grid Canada Certificate Authority (CA), an internationally recognized CA. Grid users store their credentials in a MyProxy server to be retrieved by the metascheduler or worker nodes at any time. Further, the GridX1 hosts recognize certificates signed by CA's from around the world.

The novel components developed and implemented in GridX1 are highlighted in figure 3.1 in green. A central resource broker built around Condor-G provides efficient load distribution and reliable job management. This broker discovers the GridX1 resources by periodically receiving *classified advertisements* (ClassAds) from an information provider running on the headnode of each cluster. A Condor-G JobManager has been developed to make the job scheduling and management services available as a GRAM interface. It proceeds by creating a job description file, submitting it to the Condor scheduler using condor_submit, and polling the job using condor_q. These commands act as if the GridX1 resources were in a local Condor cluster. In the case that a user's GRAM client does not delegate a proxy with full privileges along with the job, the JobManager can retrieve one from the MyProxy credential repository (see a detailed example in section 3.4.1). To accept grid jobs from Condor, each of the Linux clusters provides a GRAM interface to the local resource management system (e.g. PBS or Condor as mentioned in section 3.2). By having a GRAM client on a workstation, grid users can submit jobs to the GRAM interface on the metascheduler. Similarly, more knowledgeable users can submit jobs directly to Condor-G by using the built-in condor_submit command. The GridX1 monitoring system tracks the status of the basic grid and site services and provides a web interface for use by the grid users and operators. The following sections detail the resource management and monitoring tools in use on GridX1.

3.3.1 Resource Management and Metascheduling

GridX1 utilizes the Condor-G [25] resource management system for cluster metascheduling. The Condor system provides great flexibility in interfacing users to the GridX1 resources. Condor separates the processes which handle the management of resource advertisements, job queues, and matchmaking between jobs and resources. Additionally, Condor does not restrict system designers to a one-to-one mapping between the processes. GridX1 has made use of this flexibility to provide for a scalable

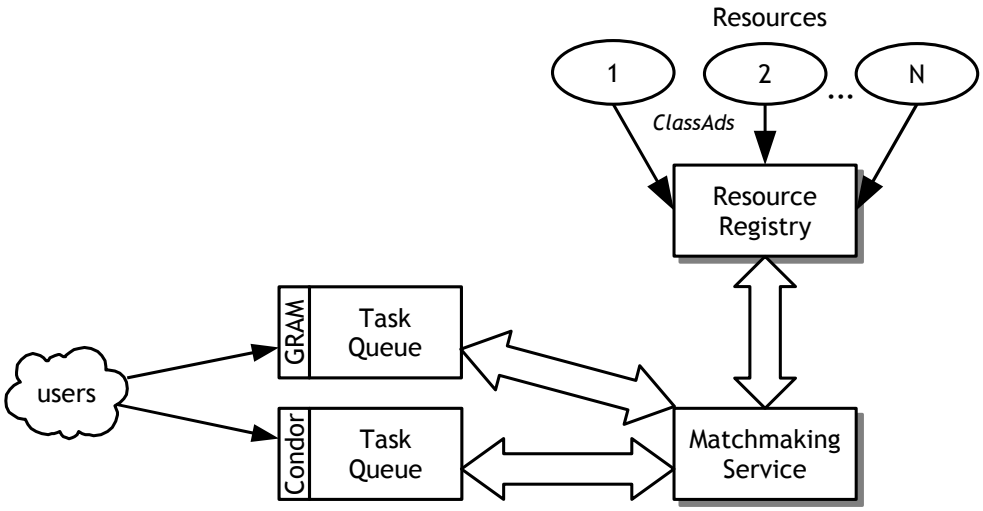

Figure 3.2: The GridX1 metascheduling architecture allows users to submit via GRAM or Condor. Cluster resources publish ClassAds to a resource registry. A matchmaking service matches jobs to appropriate cluster resources.

metascheduling architecture.

Figure 3.2 presents the metascheduling architecture of GridX1. Users can access the GridX1 metascheduler via a GRAM or Condor interface, with each of these being associated with a job queue. Though the GRAM and Condor interfaces both provide similar functionality, we incorporate both to provide flexibility to the grid users. GridX1 users may use these interfaces directly or implement and deploy their own interfaces, which might be tailored for a specific task. Cluster resources publish Classified Advertisements (ClassAds) to a resource registry. Each of the interfaces is configured to access a resource registry and a matchmaking service. The matchmaking service matches each job to an appropriate cluster resource. The flexible nature of Condor allows GridX1 to have multiple resource registries and matchmaking services. By dividing the job workload between interfaces, GridX1 can scale to large numbers of simultaneous jobs (as demonstrated in section 3.4).

Resource ClassAds: Generation and Advertisement

The Condor *ClassAd* mechanism is used to advertise resource characteristics and status to the resource registries. The ClassAd is a text file consisting of attribute-value pairs, where the attributes can be probed from each grid resource. There are several mandatory attributes in order for Condor-G to recognize the resource as a Globus-enabled cluster, rather than a single execution host. The remainder of the ClassAd is free format allowing great freedom to fully represent the cluster.

The resource ClassAd is generated by an information provider tool running periodically from a non-privileged process on the head node of each cluster. The level of customization varies from cluster to cluster, and is mainly necessary due to the differing policies implemented by the local resource management systems. The tool gathers a number of metrics from each cluster including the number of free processors, the number of jobs waiting in a queue, the number of running jobs, and the estimated time that a job will wait in a queue. After generating the ClassAd, it is published to the resource registries.

Matchmaking: Requirements and Ranks

The matchmaking service, provided by the Condor-G *collector* and *negotiator*, is responsible for matching each submitted job to an appropriate cluster resource. It accomplishes this by periodically retrieving job and resource ClassAds and evaluating the *Requirements* and *Rank* expressions therein to choose the best resource. The matchmaking process proceeds by evaluating the Rank and Requirements expressions for each pairwise combination of queued jobs and available resources. The pair having satisfied Requirements and the maximum Rank is selected, and the job is assigned to the relevant resource.

Simple and intuitive Rank expressions can be derived by incorporating metrics proportional to the number of available processors at each cluster. For example, the

```
ESTIMATE-WAIT-TIME(Q)
 1  cpus ← get_cpus_by_queue(Q)
 2  for i ← 1 to length[cpus]
 3  do
 4      if get_state(cpus[i]) = "free"
 5        then T[i] ← 0
 6        else  j ← get_job_id(cpus, i)
 7              T[i] = get_duration(j)
 8
 9  (t_min, cpu_min) ← find_min(T)
10  J ← get_jobs_by_state(Q, "queued")
11  for j ← 1 to length[J]
12  do
13      T[cpu_min] ← t_min + get_duration(j)
14      (t_min, cpu_min) ← find_min(T)
15
16  return t_m
```

Figure 3.3: The Estimate-Wait-Time algorithm.

following Rank expression for the jth job on resource i can be used to achieve a balanced distribution of jobs to each resource:

$$R_{ij} = \frac{A_i - \sum_{j=1}^{Q_i} P_{ij}}{C_i} \tag{3.1}$$

where A_i is the number of available processors, Q_i is the number of queued jobs, P_{ij} is the number of processors pending on resource i for job j, and C_i is the total number of processors. Assuming that all queued jobs are uniprocessor ($P_{ij} = 1$) the Rank reduces to:

$$R_i = \frac{A_i - Q_i}{C_i} \tag{3.2}$$

Ranking by Estimated-Waiting-Time

The simple ranking strategy presented in the previous section works well in lightly loaded grids to achieve a balanced distribution of jobs on the grid clusters. However, in practice it is found that in GridX1, where the clusters are shared facilities, there

are typically no available processors at any time. Instead users submit jobs to be queued on the cluster and executed when a processor becomes available.

To solve this problem, the concept of estimated waiting time (EWT) has been introduced into the rank expression. The purpose of the EWT is to match a job to the cluster where the job will start executing the soonest. The EWT algorithm is executed on each GridX1 resource. The EWT algorithm, shown in figure 3.3, is described below.

1. An array T is constructed with dimension n which is equal to the number of processors available to the queue Q at a resource. The value of $T[i]$ is the remaining running time for the job that is running on the ith processor as given by the queue information. The processor with the smallest value in T is identified (cpu_min) and the associated time is recorded (t_min). If there are no jobs in the state "queued" in the queue, then $T[cpu\_min]$ is used as the EWT in the ClassAd. If there are free processors and no queued jobs, then the EWT is $T[cpu\_min] = 0$.

2. If the processors are fully utilized and jobs are waiting in the queue, then EWT must include a term for these queued jobs. To calculate the EWT we evaluate a schedule that follows a first-in-first-out ordering of the jobs. We assign the first job in the queue to the processor (cpu_min) that has the smallest value in T. We correct the value of $T[cpu\_min]$ by adding the estimated duration of the first queued job.

3. The modified array T is then used to identify the next processor with the smallest time remaining (as in Step 1). The next job in the queue is assigned to this processor and its array value is corrected for this new job. This process continues until there are no jobs remaining in the queue.

4. Once there are no jobs in the queue, then the time $T(cpu\_min)$ of the processor

with the smallest value is used as the EWT.

The Rank can be expressed in terms of EWT by favouring resources with a smaller EWT:

$$R_i = -EWT - P * M_i \tag{3.3}$$

where R_i^{EWT} is the Rank of cluster i, and P is a penalty equal to the estimated mean increase in EWT from the currently matched jobs, and M_i (known to Condor-G as CurMatches) is the number of matches to cluster i in the current matchmaking cycle. CurMatches is a resource attribute that is incremented by 1 for each match made to that resource. When the resource ClassAd is refreshed it is reset to zero.

The EWT can be used along with the load balancing strategy as defined in equation 3.2 as follows:

$$R_i = max\left(\frac{A_i - Q_i}{C_i}, 0\right) + min\left(0, \frac{-EWT - P * M_i}{T_i^{avg}}\right) \tag{3.4}$$

where T_i^{avg}, the average running time of the previous jobs at cluster i, is used to make the second term dimensionless. This Rank function reduces to load balancing when there are free processors (when the EWT term becomes zero). Likewise, the EWT term becomes important when there are no free processors and the first term becomes insignificant.

Throttled Load Balancing and EWT

Even after combining the EWT with the load balancing strategy as in equation 3.4, there remain two problems that must be solved to ensure efficient allocations. First, there is a need to pre-queue some jobs on each cluster so that they start immediately when processors become available. Without pre-queueing, resource utilization is decreased because there is a delay introduced between when a processor becomes available and when the metascheduler discovers this availability. Second, it is impor-

tant not to pre-queue too many jobs, otherwise their time-limited proxies may expire and the job failure rate will increase.

A throttling technique has been used on GridX1 to overcome these problems. In this technique, jobs are submitted as quickly as possible until the local batch queues are loaded to a configurable depth. This is achieved by using the following term as the Requirements expression in the ClassAds used by the matchmaking service (i.e the following expression must evaluate to TRUE for a job and resource to be matched):

$$Q_i + M_i < f_q * C_i \tag{3.5}$$

where Q_i is the number of jobs queued at cluster i, f_q is the fraction of the total number of processors to be used as the queue size (e.g. 0.1), and M_i is the current number of matches to cluster i. Using this expression, an example site with $f_q = 0.1$ and $C_i = 100$ would be allowed only 10 jobs to be queued, after which the site would fail the Requirements. Jobs which cannot find a matching site, will remain in the queue. This technique enables the matchmaking to handle stale information sensibly and leads to a balanced job distribution amongst sites with similar Ranks.

External Data Ranking by Network Bandwidth

A novel feature of GridX1 is the introduction of external data ranking which is important when there are a lot of job failures due to network problems while reading input data from a storage facility. The rate of failures caused by this can be significantly improved by incorporating a `datarank` function which finds the computing facility which has the highest network bandwidth to a storage facility hosting the input data. The function is termed *external* because it is implemented as a shared library which is accessed at run-time by the Condor-G negotiator.

The datarank function has two prerequisites. The network bandwidth between each computing and storage facility is periodically measured and stored in a band-

width table. Also needed is a replica location service (RLS) which maps logical file names (LFNs) to physical file names (PFNs). The datarank function is defined by `datarank(lfn1, lfn2, ..., computeElement)`. During a single negotiation cycle, the function will be called with the same LFN arguments for each possible computing facility. When called, the function queries the RLS to find the PFNs corresponding the LFNs, and then consults the bandwidth table to find the bandwidth between the specified computing facility and the storage facility hosting the PFNs. The datarank function returns the highest bandwidth found. To ensure fast response times, the datarank function caches its queries to the RLS.

This function has been applied to LCG-ATLAS jobs (see section 3.4.1) and was found to be useful in significantly reducing the failure rates and thus improving the overall efficiency of the grid.

3.3.2 GridX1 Monitoring and Operations

The operation of GridX1 relies on the availability of current, accurate, and easily accessible status information. By employing a web-based grid monitoring system, users can easily track the status of resources and their jobs, and additionally, the grid operators can detect and diagnose faults on the grid.

Though there are many existing grid monitoring systems, notably the Inca Test Harness [49] and MonALISA [50], a new monitoring system was developed to meet the specific needs of GridX1 without incorporating the unneeded features from the other projects. Since the GridX1 metascheduler is built around Condor-G, the monitoring system needs to interface with the Condor queue, pool status, and history functions in order to show the GridX1 resource and job status.

The GridX1 monitoring system (figure 3.1) is built around a central daemon which periodically evaluates the health of the grid. The middleware status of each cluster resource is determined by performing an authentication test, a GRAM job submission test, and a GridFTP data transfer test. The results of these tests are archived in a

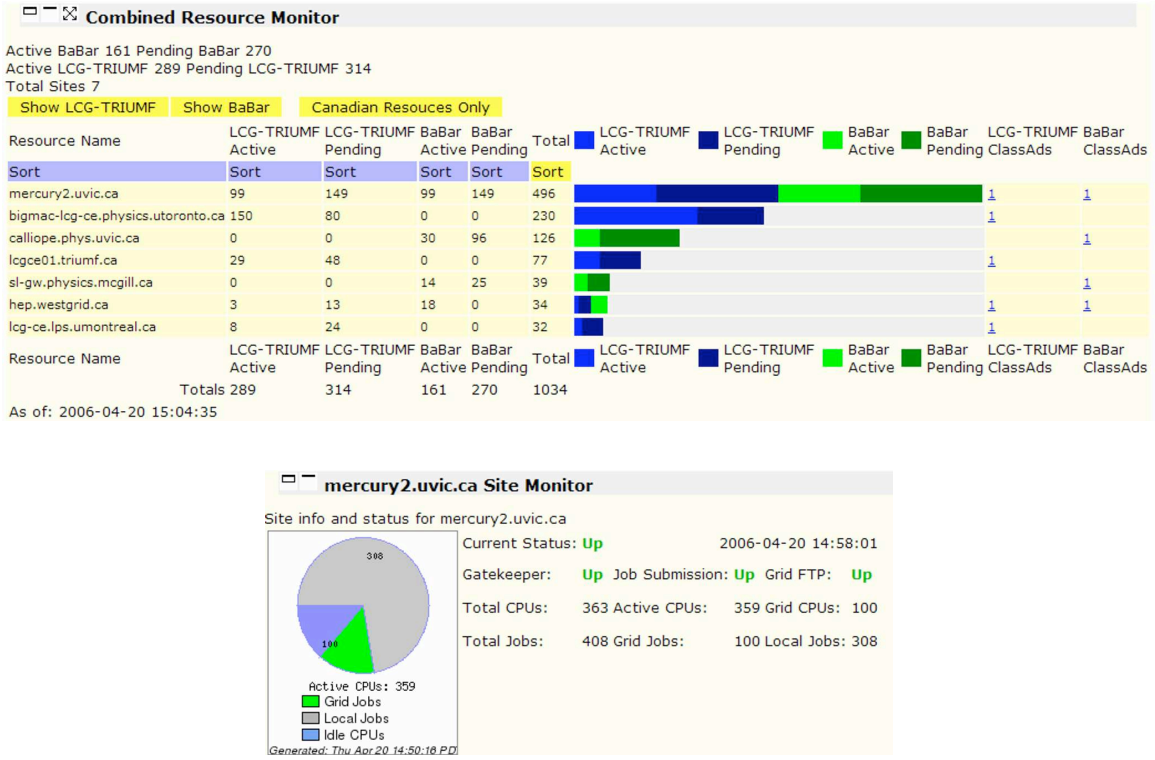

Figure 3.4: A web-based monitoring system presents the current job and resource status including a resource summary table showing the number and type of jobs running on each cluster (upper), and the detailed queue and grid service status for a specific cluster (lower).

database.

A web-interface to the monitoring system presents grid status information in an easily accessible form. Figure 3.4 (upper) shows a table and bar chart which tracks the resource usage and availability at each cluster as advertised to the resource registry. Also, the pie chart in figure 3.4 (lower) shows the current number of grid and local jobs running, along with the recent GRAM and GridFTP test results. Additionally, the current status of each Condor job queue is displayed in table form, and the overall grid health is summarized in a national map (both not shown here for conciseness). Cluster-level monitoring is provided by Ganglia [51] at most of the sites.

Figure 3.5: The GridX1 resources are federated into the LCG, with all GridX1 resources appearing as a single LCG computing facility.

3.4 User Applications on GridX1

3.4.1 ATLAS and the LCG

GridX1 has been used extensively for the ATLAS particle physics simulation application, notably during the Data Challenge 2 (DC2) [52] in 2004 and the Rome Production in 2005. DC2 and Rome were large scale tests of the international ATLAS grid computing system. The ATLAS experiment is one of four major projects on the LHC at the CERN Laboratory in Geneva. Many collaborators are contributing computational resources to the ATLAS experiment by providing dedicated resources to the LCG. In order to meet the Canadian computing requirement for the ATLAS experiment, we have developed an alternative approach which federates the GridX1 resources into the LCG while maintaining their shared nature.

Grid federation is the process by which the resources of multiple compute grids are combined to form a single larger grid. One method that can be used to federate grids is a grid-nesting model, in which child-grids appear to be virtual clusters within a parent-grid. A virtualization layer at the root of each child-grid implements all of the

functions of a real cluster, including job submission, status polling, and cancelling. In the case of job submission, the virtualization layer re-submits the job to the child-grid.

In order to make the GridX1 resources available to the LCG, an interface was developed to implement the virtualization layer [53]. The federation of GridX1 resources into the LCG is exhibited in figure 3.5. The interface to LCG is provided by an LCG Compute Element, with a Globus JobManager of type Condor-G (as described in section 3.3) to interface to the GridX1 resource management system. Also provided is an information provider which advertises the combined availability of all the GridX1 resources to the LCG.

Since the re-submission of jobs to the GridX1 clusters proceeds via the GRAM protocol, it requires a *full user proxy*. The proxy that arrives with the job from the LCG Resource Broker is *limited* in the sense that it can be used for GridFTP transfers, but not for a further GRAM submission. Full proxy delegation via the GRAM protocol is possible, however this may result in security concerns. Alternatively, a shared full proxy that can be accessed from the interface machine could be used, but this would limit usage accountability. The implemented solution is to utilize the MyProxy online credential repository. This service is used by the Condor-G JobManager to delegate a full proxy using a limited proxy to authenticate.

During the ATLAS Data Challenge 2 in 2004 and Rome production in 2005, the GridX1 system has been quite effective in both execution efficiency and ease of maintenance. The number of successful ATLAS jobs executed on GridX1 between July 2004 and June 2005 exceeded 20000, with more than 90% of these being executed between March and June 2005. Additionally, the success rate of jobs on GridX1 was similar to that of the entire LCG at approximately 50%. This seemingly high failure rate was due to instabilities in the LCG as a whole (90% of the failures were due to the unavailability of a few storage facilities), and rarely did GridX1-specific issues cause problems. Cluster utilization has remained high, with usage loads peaking at

over 200 simultaneously running jobs. Finally, as mentioned in section 3.3.1, the usage of the external datarank function was found to eliminate the problems related to the unavailable storage facilities, reducing the error rate by 90%.

3.4.2 BaBar Monte Carlo

The BaBar experiment at the Stanford Linear Accelerator Center studies the asymmetry between matter and antimatter in the Universe. In parallel with the physical detector, the BaBar Monte Carlo (MC) application generates synthetic particle collisions, which are used for consistency checking and physics analysis. The Canadian computational contribution to the BaBar experiment has typically come from a number of individually managed cluster resources. Recently, the BaBar MC application has been adapted to execute on GridX1. As a result, the application management is simplified by collapsing multiple managers to a single interface. Further, utilizing GridX1 provides the application with access to new resources which will result in increased production rates.

The process of grid-enabling the BaBar MC application requires the installation of the application package at each GridX1 resource. This includes a local Objectivity database server containing a conditions database and background triggers. A submit workstation is configured to act as the BaBar MC grid management node using Condor-G, and it handles application specific jobs such as building and creating tar archives of the input execution directories. When a job is submitted to the GridX1 metascheduler (depicted in figure 3.2), the matchmaking process selects an appropriate resource for the execution of the job. Once the job is finished, a tar archive of the output is transferred back to the submit workstation. The output data from all the sites is merged and exported to the production database at SLAC. A web-based monitor presents the status of the BaBar production jobs on GridX1 (figure 3.4).

The results of this activity have been successful for the BaBar community. The

approach presents a computing solution that allows the production rates to increase without substantial increases in administration costs.

3.5 Conclusions

The architecture of GridX1 emphasizes standards compliance, by building around the de-facto standard Globus Toolkit, and ease-of-deployment, by leveraging proven technologies such as the Virtual Data Toolkit and Condor-G. The incorporation of Condor's matchmaking system into the GridX1 metascheduler has allowed for reliable operation and provided a mechanism for improving the resource utilization. A variety of Ranking strategies were presented, starting from a simple load balancing strategy and later incorporating estimated wait time and job pre-queueing and throttling. These strategies are used in production to provide a scalable and efficient allocation of resources on GridX1. Further, a novel external data ranking function was proposed and implemented to decrease job failure rates due to network problems. This was used for the ATLAS application to greatly decrease the job failure rate. Web-based monitoring information has proven to be useful to grid operators and users alike. By monitoring the basic grid functionalities, problematic clusters are easily identified and can be removed from the pool of available resources to enhance the reliability and efficiency of the grid.

The ATLAS and BaBar particle physics applications have proven to be successful in exploiting the GridX1 infrastructure and resources. The result of running these applications on GridX1 has shown an increased production capacity while using the same man-power commitment. Other applications can be easily adapted to run and make use of the compute resources available on GridX1. Finally, by developing a method to federate GridX1 resources with the LCG, we have made a large computational contribution to the ATLAS experiment while using shared facilities that would be otherwise unavailable.

Chapter 4

Grid Resource Allocation using a Utility Model and Knapsack Formulation

Resource allocation is one of the central challenges in computational grids. In this chapter, we introduce a novel strategy for the allocation of grid resources. Using a utility-model approach, a technique which evaluates the usefulness or value of each allocation option, allocation preferences can be described objectively. By then formulating the allocation as a knapsack problem, the grid resources can be allocated according to the preferences described by the utility-based heuristics. The utility/knapsack approach is evaluated in a simulation using a variety of task workloads; in each case the strategy is shown to perform allocations congruent with the defined allocation preferences.

The work presented in this chapter was published in the Proceedings of Grid'04, the Fifth IEEE/ACM International Workshop on Grid Computing [54], and the Proceedings of HPCS'06, the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment [55], and has been submitted for publication in the Future Generation Computer Systems journal.

4.1 Introduction

Allocating resources on computational grids is a complex procedure involving coordinated resource sharing and meeting the requirements of users and resource owners. Notable previous research, described in detail in chapter 2, includes backfilling [17], makespan minimizing heuristics [18, 19, 20, 21], gang scheduling [22], matchmaking [23, 24, 25], and economic approaches [26, 56, 27, 28, 57].

In this chapter, we improve upon the existing research by introducing a novel grid resource allocation strategy which is composed of a *utility model* (UM) [31] and a formulation of the allocation as a knapsack problem. The UM-approach is used to objectively describe the grid resource allocation options; it is developed from the idea that the allocation options vary in their perceived value, or utility, to a user or resource owner. Using the UM, the allocation problem can be naturally formulated as a variant of the 0-1 multichoice multidimensional knapsack problem (MMKP); the MMKP is used to find the set of allocation options that optimizes the global utility sum. Using this combined utility/knapsack approach, Quality of Service (QoS) measures can be objectively defined and implemented on the grid.

Resource allocation strategies can be classified according to two criteria [58]. The first criterion involves the description of the grid, which can be either state-based or model-based. The second criterion involves task behaviour; in a preemptive environment, tasks assigned to hosts are allowed to be paused and migrated to other hosts, while in a non-preemptive environment, tasks are required to perform all of their execution on a single host.

In this work we assume a state-based preemptive form of a grid. That is, there exists a mechanism through which resource states and task metadata can be identified. Tasks are assumed to be checkpointable and malleable, meaning that tasks can be paused to be migrated between processors and that the number of processors used

in parallel can be varied.

Whereas a significant number of grid applications are neither checkpointable nor malleable, indications are that grid deployments will soon incorporate system software that make these assumptions realistic. For example, by building a grid using virtual machine technologies such as Xen [59], the grid will allow application images to be easily paused and migrated to a new host. Additionally, the independent nature of many grid applications allows malleability to be achieved by executing more than one instance of an application on a single host. For example, a 64 process task can be executed on 32 processors by submitting 2 processes to each processor, if memory limits permit. It is important to note that the techniques presented in this chapter are also applicable as an admission control system in non-checkpointing environments. In such grids, the system will use the utility/knapsack approach to optimize the initial allocation of each task. However, without checkpointing, newly arriving high priority tasks will not be able to preempt lower priority tasks; thus, the system will not achieve the same overall performance as when checkpointing is available.

The remainder of this chapter is organized as follows. In section 4.2 we introduce the notion of utility in relation to grid resource allocation. Next, in section 4.3 we formulate resource allocation on the grid as a variant of the 0-1 multichoice multidimensional knapsack problem. We then introduce allocation policies and their utility functions (section 4.4) and demonstrate these strategies using random and real task workloads (section 4.5). Finally, in section 4.6 we reflect on our findings and conclude.

4.2 Grid Resource Allocation and Utility

A computational grid is comprised of many services; the service focused on in this dissertation is the job management service, or, more specifically, the resource broker. The primary function of the resource broker is to match user-submitted tasks to the

grid resources in a manner which is beneficial to both users and resource owners.

When a grid task is being submitted to a job management service, it is accompanied by a description of its required resources, such as the number of processors, the amount of memory and storage, and its estimated computation length. In addition, each task lists a number of allocation options, each of which reflects varied resources requirements resulting from task's malleability and the different resources on the grid. Each of the option's associated measures, known as the QoS metrics, are used to compute a *utility* value.

The utility value is a measure of the usefulness of an allocation option to the owner of both the tasks and the grid resources. As such, it depends on metrics of the computation task itself (including the length of the computation, the accumulated computation time, or the existence of a completion deadline) but also on metrics that directly reflect the perceived value that the owner of the task attaches to the task itself.

For example, consider a task which requests that it be run on a cluster of processors and lists two options:

1. requires 8 processors and has a user-assigned value of 100 units.
2. requires 1 processor and has a user-assigned value of 10 units.

Note that the example options mentioned above could be structured to include more resource types, such as memory sizes, interconnect bandwidth, and file size; the allocation of multiple resource types is the subject of chapter 6. Because option one uses more processors, it will likely complete the execution more rapidly, and is thus more desirable to the user. The higher value assigned to the first option reflects this desirability, and the resource broker will include this metric in the computation of the option's utility. Having a set of task options and their computed utility values, the resource broker is then required to allocate resources to tasks in a manner that is close to optimal.

There are several algorithms that could be used to perform the resource allocation. In this dissertation, we present the resource allocation problem as a variant of the 0-1 multichoice multidimensional knapsack problem, and solve the problem using heuristics [60, 34].

4.2.1 Utility of a Task Option

In this work, utility values are used as coefficients in an objective function which is composed of the sum of selected option utilities. The utility represents the relative merits of each option and depends on QoS metrics associated with each option. The aim of resource allocation is to find the set of options that will maximize the objective function while observing resource constraints. A utility function U is defined and used to compute the utility values from the QoS metrics.

QoS metrics may be intrinsic to the option, being calculated based on intrinsic properties of the task under the option, or external to the option, being assigned by the user. Intrinsic options may include the task's computation length, accumulated running time, deadline, or a number of other measures.

An external metric usually reflects the option's perceived value to the owner of the task. This value cannot be assigned arbitrarily by the user; rather each user is allocated a maximum credit which can be portioned out to each of their tasks. An option which is deemed to be highly desirable (e.g. an option that will result in a fast response time for an urgent task) may be given a large portion of the available credit while a less desirable option may be given a smaller portion.

The utility value for option j of task i is denoted by u_{ij} and is obtained by the evaluation of the utility function $U(x)$. In general, $U(x)$ is a monotonically increasing function of the QoS-metric-parameter P_{ij}, i.e. $u_{ij} = U(P_{ij})$. The QoS-metric-parameter P_{ij} is computed from the intrinsic and external QoS metrics p_{ijl};

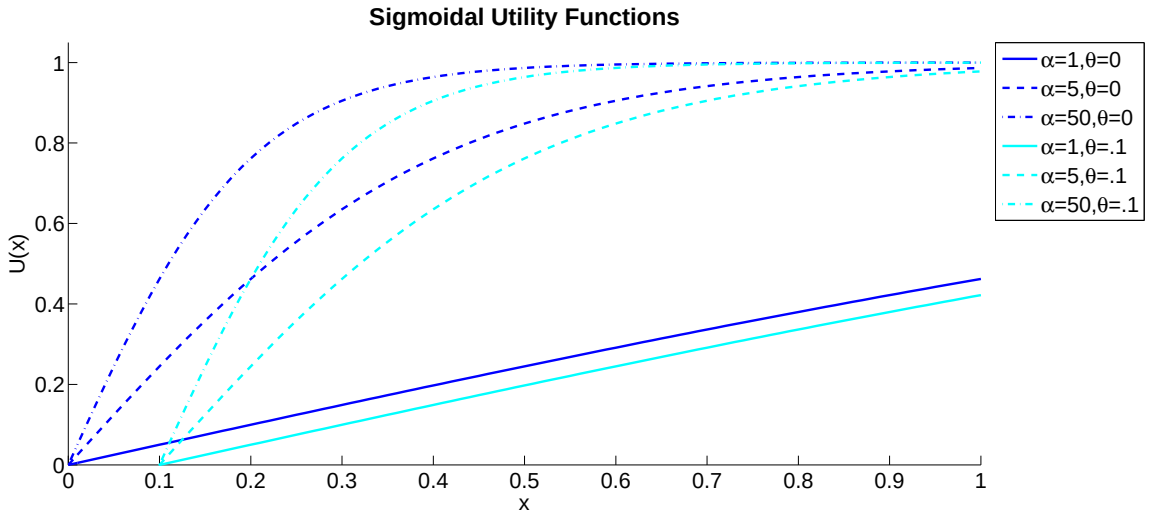

Figure 4.1: Example sigmoidal utility functions of the form $U(x) = \frac{2}{e^{-\alpha(x-\theta)}+1} - 1$.

$l = 1 \ldots m_{ij}$ as

$$P_{ij} = \frac{\sum_l w_l S(p_{ijl})}{\sum_l w_l} \tag{4.1}$$

where w_l are non-negative weighting factors and $S(x)$ is a non-decreasing positive normalizing function. Varying the weights w_l in the the QoS-metric-parameter P_{ij} different policies can be implemented. For example if $p_{ij1} = 1/T_i$ and $w_1 = 1$, then tasks with short run times are preferred (assuming that T_i denotes the run time of task i).

An example of a saturating utility function, depicted in figure 4.1, is

$$U(x) = \begin{cases} 0 & \text{when } 0 \leq x \leq \theta \\ \\ \dfrac{2}{e^{-\alpha(x-\theta)}+1} - 1 & \text{when } \quad x > \theta \end{cases} \tag{4.2}$$

where α and θ affect the sharpness and offset of the function, respectively. The same function can be used as the normalizing function $S(x)$. Note that other utility and normalizing functions can be considered, including the identity function $I(x)$.

4.3 The Knapsack Formulation

The UM-approach to allocation option evaluation requires an allocation technique which can find a nearly optimal mapping of tasks to resources. In this section, one such solution is formulated as a knapsack problem. First, an example of a UM-based resource allocation formulation is presented, then the general formulation is introduced.

4.3.1 An Example

Assume that a grid is comprised of a number of clusters (usually clusters of processors). R_k represents the number of processors available in cluster k, a_{ijk} represents the required number of processors from cluster k by task i as per option j. Similarly, variable $x_{ij} \in \{0, 1\}$ represents whether task i was allocated option j ($x_{ij} = 1$) or not ($x_{ij} = 0$).

Assume that a certain grid is comprised of two clusters: *Cluster1* and *Cluster2*. *Cluster1* includes R_1 fast processors while *Cluster2* includes R_2 slower processors. At time t, two new tasks, *Task1* and *Task2*, request resources from the grid. Each task has a number of options listed. In this example, the only resource that is to be allocated is processors, and the utility value associated with each option is calculated from an external QoS metric V_{ij} assigned by the owner of the task.

Thus, *Task1* is submitted with the following three options:

- option 1 requires a_{111} processors from *Cluster1*. The user assigns an external value V_{11} to the QoS metric p_{111} associated with this option, thus $p_{111} = V_{11}$.
- option 2 requires a_{121} processors from *Cluster1*. The user assigns an external value V_{12} to the QoS metric p_{121} associated with this option, thus $p_{121} = V_{12}$.
- option 3 requires a_{132} processors from *Cluster2*. The user assigns an external value V_{13} to the QoS metric p_{131} associated with this option, thus $p_{131} = V_{13}$.

while *Task2* lists the following three options:

- option 1 requires a_{211} processors from *Cluster1*. The user assigns an external value V_{21} to the QoS metric p_{211} associated with this option, thus $p_{211} = V_{21}$.
- option 2 requires a_{222} processors from *Cluster2*. The user assigns an external value V_{22} to the QoS metric p_{221} associated with this option, thus $p_{221} = V_{22}$.

Given the above description of the proposed options, one can compute the utility value associated with each option as $u_{ij} = U(S(p_{ij1}))$ since only one external QoS metric was involved. Accordingly, we can formulate the constrained optimization problem as:

$$\begin{aligned}
&\text{Maximize } f(x) = \; u_{11}x_{11} + \; u_{12}x_{12} + \; u_{13}x_{13} + \; u_{21}x_{21} + \; u_{22}x_{22} \\
&\text{Subject to} \qquad \begin{aligned}
a_{111}x_{11} + a_{121}x_{12} \qquad\qquad + a_{211}x_{21} \qquad\qquad &\leq R_1 \\
a_{132}x_{13} \qquad\qquad + a_{222}x_{22} &\leq R_2 \\
x_{11} + \quad x_{12} + \quad x_{13} \qquad\qquad &= 1 \\
x_{21} + \quad x_{22} &= 1
\end{aligned} \\
&\qquad\quad x_{ij} \in \{0, 1\}
\end{aligned} \tag{4.3}$$

It is clear that finding the optimal allocation is relatively simple in this example. However, when a large number (thousands) of tasks and resources have to be considered, then the selection of the algorithm to solve the knapsack problem has ramifications for the speed and accuracy of the solution [61, 62, 60, 34]. This study employs the heuristic approach described in [33]. Being a heuristic, the approach does not guarantee that the solution found is optimal. However, this technique has been chosen for providing solutions that are very close to optimal within a reasonable amount of time.

4.3.2 General Formulation

The formulation of the 0-1 knapsack problem for allocation of grid resources of a single type is given as:

The overall utility of the selected task options is optimized by maximizing the objective function $f(x)$. Each task i (of the n in total) has m_i options. The utility

$$
\begin{aligned}
\text{Maximize} \quad & f(x) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} u_{ij} x_{ij} \\
\text{Subject to} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m_i} a_{ijk} x_{ij} \leq R_k \text{ for } k \in \{1, .., r\} \\
& \sum_{j=1}^{m_i} x_{ij} \leq 1 \text{ for } i \in \{1, .., n\} \\
& x_{ij} \in \{0, 1\}
\end{aligned} \tag{4.4}
$$

of the jth option of i is u_{ij}. The resource demand of option j on cluster k (of the r clusters in total) is given by a_{ijk}, and is limited to the total amount of resources available R_k at the kth cluster. x_{ij} equates to 1 or 0 when an option is selected or not, respectively.

Tasks are scheduled by periodically performing a reallocation using the knapsack formulation. Reallocations are used to re-optimize the objective function, as the utility values u_{ij} vary over time with the changes in the QoS metrics. By evaluating the utility of all task options in the queue, the optimal resource allocation is obtained. Frequent reallocations result in more optimal schedules, however, preemption overhead eventually negates the efficiency of the schedule. One strategy that can be used to increase the efficiency without frequent reallocations is to incorporate backfilling of idle processor time with short tasks.

4.4 Allocation Policies and Utility Heuristics

The utility/knapsack approach to grid resource allocation presents a general framework for describing and implementing allocation policies. The large variety of grid deployments in practise leads to a wide range of desired allocation policies. For example, grids with economic interests may favour profit, whereas altruistic grids may favour timeliness. In this section, we present a number of basic allocation policies and their associated utility function heuristics.

4.4.1 Pure Utility-based Allocation (UM)

The pure-UM approach to resource allocation uses only the utility/knapsack formulation presented in the previous sections. Resources are allocated at set scheduling time slices τ (e.g. every 12hrs). Tasks that have accumulated during the previous allocation interval together with tasks that are checkpointed at the end of the previous allocation interval participate in the allocation process. Each task offers a number of options, and the allocation policy formulates an objective function and chooses the options that will maximize this function with the resource constraints. We have experimented with a number of utility function heuristics reflecting policy choices. These are as follows:

Policy 1: Credit-value

Under this policy, the user is allocated a fixed number of credits which are managed by allocating a differing amount to each of the tasks being submitted. Credits loosely represent the value of the computation. Thus, a more valuable computation task is awarded a higher credit-value as compared to a less valuable one. Similarly, a faster computational option is attributed a higher credit as compared to a slower one. If we denote by v_{ij} the credit-value assigned to task i under option j, then the utility value u_{ij} is simply

$$u_{ij} = v_{ij} \tag{4.5}$$

Observe that the utility function used here is simply the identity function. Also, the credit-value metric assigned to the different computational options for a given task typically varies non-linearly with the computational capability of the option considered. This technique is employed by users to ensure that a single fast option is preferred over multiple slower options; for example, a 64 process task should be valued more highly than two 32 process tasks. For the experimentation reported in this dissertation, we have made the following assumption: for task i under options

m and n and if P_{im} and P_{in} are the number of processors used under the respective options, then

$$\frac{v_{im}}{v_{in}} = (\frac{P_{im}}{P_{in}})^{\alpha} \tag{4.6}$$

where α is the factor of nonlinearity and is usually greater than 1. Note that different nonlinear expressions could be chosen depending on the grid environment. In practise, users often have a minimum performance that is deemed acceptable; for example, a malleable task is infeasible below a certain number of processors, and a user would therefore assign a value of zero to options below the threshold. In the model presented in this work, this bounding on the assigned values is achieved by simply limiting the set of options available to a task to those that are feasible and acceptable.

Policy 2: Credit-value mediated by Estimated Response Time

In this policy, each task and computational option is assigned a number of credits as per policy 1. However, an additional factor is introduced to reflect the estimated completion time of the task under the option considered. This factor is included in order to ensure that highly-valued tasks do not occupy all of the available resources (i.e. small-value tasks will still complete in a timely manner). We calculate the normalized estimated task completion time c_{ij} of task i under option j as

$$c_{ij} = (t - s_i + r_{ij})/N_i \tag{4.7}$$

where s_i is the submission time for task i, t is the present time, r_{ij} is the estimated remaining computation time for task i under option j, and N_i is the computation time of task i if it were to run on a standard notional computational resource. The utility value for task i under option j is then simply

$$u_{ij} = v_{ij}\frac{t - s_i + r_{ij}}{N_i} \tag{4.8}$$

Under such a formulation, tasks which wait for long periods of time slowly increase their utility value and eventually they get the opportunity to run.

Policy 3: Credit-value mediated by Estimated Response Time with Sigmoidal Utility Function

This policy employs the credit-value and estimated response time metrics used in policy 2, yet uses a sigmoidal utility function rather than the identity function used previously. The utility function used for policy 3 is

$$U(x) = \frac{2.0}{\exp(-0.5x) + 1} - 1 \tag{4.9}$$

and the utility value is

$$u_{ij} = U\left(v_{ij}\frac{t - s_i + r_{ij}}{N_i}\right) \tag{4.10}$$

Policy 4: Estimated Response Time with Nearness to Completion Time

This policy introduces two variations. Namely, the credit-value metric is excluded, and it adds a term that raises the priority of tasks which are close to completing. The nearness to completion time (NTCT) metric is introduced in order to reduce the number of tasks that have little computation remaining. Using the previously introduced notation, NTCT is the ratio of the total computational time N_i over the remaining time r_{ij}. Thus, the utility of task i under option j is

$$u_{ij} = \frac{t - s_i + r_{ij}}{N_i} + \frac{N_i}{r_{ij}} \tag{4.11}$$

Under this policy, as r_{ij} becomes smaller, the utility becomes larger and it becomes unlikely to yield to another task.

Policy 5: Estimated Response Time with Closeness to Completion Time with Sigmoidal Utility Function

This is a variation on policy 4 which utilizes the sigmoidal utility function given in equation 4.9 instead of the identity function.

4.4.2 Combined Utility-based and Backfilling Allocation (UM+BF)

When the UM approach is used, resources which become available between allocation cycles sit idle until the next allocation period. This can lead to a significant decrease in the grid's efficiency. However, by introducing a backfilling strategy [17] between the allocation periods, utilization can be improved while maintaining the QoS results obtained by the pure UM approach.

For the UM+BF approach, we have utilized the same policies and utility function heuristics as in the UM approach, and the allocation is supplemented by backfilling between allocation cycles.

4.4.3 Reference Strategies

A number of existing allocation strategies were described in chapter 2. For reference purposes in evaluating the utility/knapsack strategy, we have implemented the first-come-first-served (FCFS), pure backfilling (BF), min-min heuristic (m-m), and gang scheduling (Gang).

4.5 Experimentation via Simulation

Using a simulated computational grid, we have performed an evaluation of the performance of the UM and UM+BF strategies using the five policies describe in section 4.4 in comparison to the reference strategies. We have experimented with two different task populations; the first has been generated randomly according to a function described in section 4.5.1, while the second is based on actual loads made public by NPACI [63]. The histogram of task lengths for the two populations are depicted in

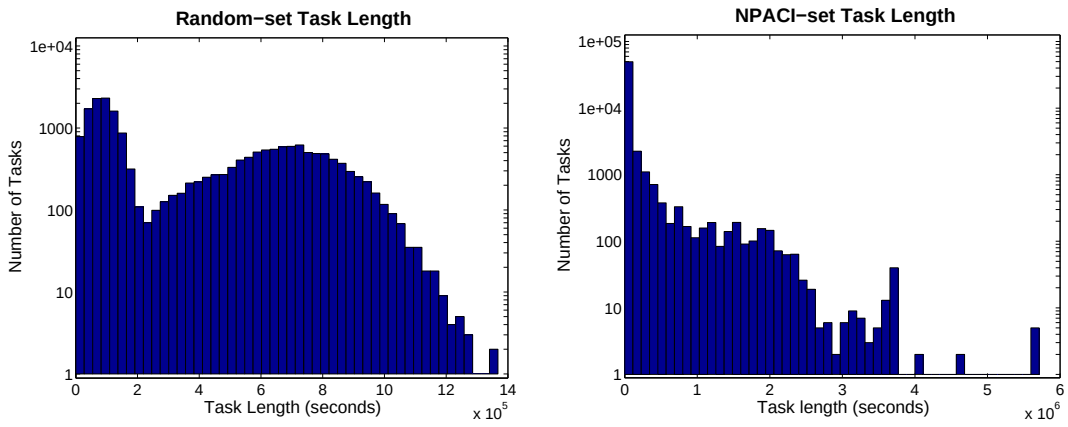

Figure 4.2: Histograms of the task computation lengths found in a randomly generated workload (left) used in section 4.5.1, the NPACI JOBLOG (right) for the study in section 4.5.2. Note that the axis ranges are different for the two histograms.

figure 4.2. The significance of using the second population is that the real workload is characterized by long-tailed distributions, which inevitably affect the performance of the various allocation policies discussed. We first present results based on the randomly-generated tasks population, and subsequently we present and compare results based on the NPACI-derived task population.

4.5.1 Experimentation with a Random Workload

The allocation strategies described in this chapter have been first evaluated using a randomly generated task workload. This random workload is useful because it presents an environment that is not complicated by the anomalies of real workloads, such as their statistical long tails. The task lengths in the random workload are depicted in a histogram in figure 4.2 (left). This is contrasted with the long-tail of the real, NPACI-derived, workload shown in figure 4.2 (right). For the random workload, task arrivals are Poisson with mean interarrival time λ, which is varied for the experiments to follow. The computation demand for each task follows a bimodal distribution function. This demand is expressed as the required computation time on a standard notional computational resource (i.e. a single processor). Each lobe of the distribution is a Gaussian with mean and standard deviation as follows. Short

tasks have a length mean value of 1 day with a standard deviation of 12hrs. Long tasks have a length mean value of 8 days with a standard deviation of 2 days. Fifty percent of the tasks are short tasks. The maximum credit value associated with each task is normally distributed around 10 units with a standard deviation of 4 units; the values of the varied options are computed using equation 4.6.

Note that throughout this study there is assumed to be no difference between a task's execution time estimated a priori and its actual execution time. This is equivalent to considering the execution time to be an advanced reservation of a resource. In such a case, a task which finishes earlier than its estimated completion time results in a temporarily idle processor; a task which outruns is estimated completion time is killed and must therefore be resubmitted with a more accurate completion time.

Experimental Setup

The randomly generated workload is evaluated in simulation of a computational grid comprising four clusters with 4, 8, 16 and 32 processors respectively. All simulations used in this dissertation were developed using the SimGrid toolkit [64]. The cluster processors are assumed to be homogeneous, to simplify the environment. A job management service accepts the task workload and allocates to the grid processors using the policies described in section 4.4.

Each experiment is comprised of a sequence of 2000 tasks; we use the average results of ten such experiments to draw conclusions. The confidence intervals shown in this dissertation represent the standard deviation on the mean, which is defined as the standard deviation of a normal distribution divided by the square root of the number of experiments [65]. For each task, we generate a number of options by varying the requested number of processors and assigning computation times and credit-value metrics that were related to the requested number of processors as per equation 4.6, with $\alpha = 1.1$. The utility values for the task options are calculated using the utility function u_{ij} corresponding to each policy.

Results of the Random Workload

For each experimental sequence, we record its overall completion time (i.e. the time required to complete the entire sequence of tasks), as well as the submission s_i and actual completion C_i times for each task i in the sequence and calculate the response times as $R_i = C_i - s_i$. We further consider the performance of the FCFS policy as the baseline, to which the performance of the proposed policies is compared. To accomplish this, we calculate the speedup achieved on a per task basis by dividing the response F_i of task i under the FCFS strategy with the response R_i achieved by each of the proposed policies. We use the per-task speedup metrics and correlate these to the per task assigned credit-value metric. The correlation value of two random variables X and Y is computed as:

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_X s_Y} \tag{4.12}$$

where $\bar{x}$ is the mean value of x, s_X is the standard deviation of x and n is the number of measurements. Further, we observe the startup delay of each task with respect to the credit-value assigned. Tables 4.1, 4.2 and figures 4.3 and 4.4 present the results of the simulations; data is present as $\mu \pm \sigma_\mu$, where μ is the mean and σ_μ is the error on the mean over the experiments. Full results are included in Appendix 1. Table 4.1 summarizes the overall completion times for each allocation strategy at varied task interarrival times λ; results are shown for reallocation period τ = 6hrs. The complete results for table 4.1 is included in table A.1; results for all λ and τ values are presented there. Table 4.2 tabulates the correlation of credit-value with per-task speedup (relative to the baseline FCFS strategy). The complete results for this table are available in A.2. Finally, figure 4.3 shows scatter plots of the speedup versus credit-value for individual tasks, while figure 4.4 depicts scatter plots of the startup delay versus credit-value.

Table 4.1: Overall completion time (units of 10^7s) for the random workload and $\tau = 6hr$

Strategy	$\lambda = 0$hr	$\lambda = 1.92$hr	$\lambda = 6.4$hr
FCFS	1.60 ± 0.02	1.62 ± 0.02	4.70 ± 0.03
BF	1.37 ± 0.01	1.48 ± 0.01	4.70 ± 0.03
m-m	1.39 ± 0.01	1.48 ± 0.01	4.70 ± 0.03
Gang	1.93 ± 0.04	1.52 ± 0.01	4.71 ± 0.03
UM Pol.1	1.44 ± 0.01	1.48 ± 0.01	4.70 ± 0.03
UM Pol.2	1.45 ± 0.01	1.48 ± 0.01	4.70 ± 0.03
UM Pol.3	1.39 ± 0.01	1.47 ± 0.01	4.70 ± 0.03
UM Pol.4	1.45 ± 0.01	1.48 ± 0.01	4.70 ± 0.03
UM Pol.5	1.41 ± 0.01	1.48 ± 0.01	4.70 ± 0.03
UM+BF Pol.1	1.34 ± 0.01	1.47 ± 0.01	4.70 ± 0.03
UM+BF Pol.2	1.34 ± 0.01	1.47 ± 0.01	4.70 ± 0.03
UM+BF Pol.3	1.34 ± 0.01	1.47 ± 0.01	4.70 ± 0.03
UM+BF Pol.4	1.37 ± 0.01	1.47 ± 0.01	4.70 ± 0.03
UM+BF Pol.5	1.36 ± 0.01	1.47 ± 0.01	4.70 ± 0.03

Discussion of the Random Workload Results

In order to compare the schedule efficiency, table 4.1 lists the mean overall completion times for the investigated strategies. Results in the summary table are presented for $\tau = 6$hr and $\lambda = 0$hr, 1.92hr, and 6.4hr, which we will refer to as heavily loaded, moderately loaded, and lightly loaded cases, respectively. Under heavy loading, the lowest overall completion times result from the UM+BF policies 1, 2, and 3. From this we can conclude that the combination of backfilling with the utility/knapsack approach leads to the fastest schedules. Also, we observe that the policies that employ the sigmoidal utility function (policies 3 and 5) perform better than their non-sigmoidal counterparts (policies 2 and 4). We conclude that the compression of the utility values is reducing the negative impacts of outlying options.

Under moderate loading, the performance of all strategies, except FCFS and Gang becomes similar, and under light loading we see that all strategies lead to the same overall completion time. At very long task interarrival times the computational resources are under-utilized, and as such every task is allocated as soon as it arrives At these light loads, the overall completion time depends only on the arrival of the

Table 4.2: Correlation of credit-value with speedup for the random workload and $\tau = 6hr$.

Strategy	$\lambda = 0$hr	$\lambda = 0.64$hr	$\lambda = 1.6$hr
BF	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01
m-m	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01
Gang	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
UM Pol.1	0.24 ± 0.01	0.16 ± 0.01	0.07 ± 0.01
UM Pol.2	0.10 ± 0.01	0.10 ± 0.01	0.07 ± 0.01
UM Pol.3	0.12 ± 0.01	0.11 ± 0.01	0.08 ± 0.01
UM Pol.4	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01
UM Pol.5	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01
UM+BF Pol.1	0.23 ± 0.01	0.13 ± 0.02	0.06 ± 0.01
UM+BF Pol.2	0.10 ± 0.01	0.09 ± 0.01	0.06 ± 0.01
UM+BF Pol.3	0.12 ± 0.01	0.09 ± 0.02	0.06 ± 0.01
UM+BF Pol.4	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01
UM+BF Pol.5	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01

last task and its required computation time.

One of the main principles of the utility/knapsack strategy is the use of a credit-value metric to allow users to intervene and prioritize urgent tasks. The effectiveness of the varied policies in implementing this is shown in table 4.2, which tabulates the correlation of option credit-value with per-task speedup as compared to the FCFS strategy. A positive correlation is evident for UM and UM+BF policies 1, 2 and 3; each one of which includes in its utility function a factor corresponding to the credit-value assigned to the task. This means that indeed the assignment of a credit-value can be used to allocate tasks preferentially without diminishing the overall efficiency of the scheduling policy. Note that the completion time is not negatively affected; indeed it is improved as compared to the reference strategies, and when the UM policies are combined with backfilling, they provide optimum completion times.

Figure 4.3 provides further insight as to the ability of the UM policies to complete high valued tasks earlier. Figures 4.3 (a) and (c) show the behaviour of the reference Gang and BF policies. It is evident that there is no correlation between the credit-value and the speedup of the tasks. On the other hand, figures 4.3 (b) and (d) depict the behaviour of the UM and UM+BF strategies, where evidently the high valued

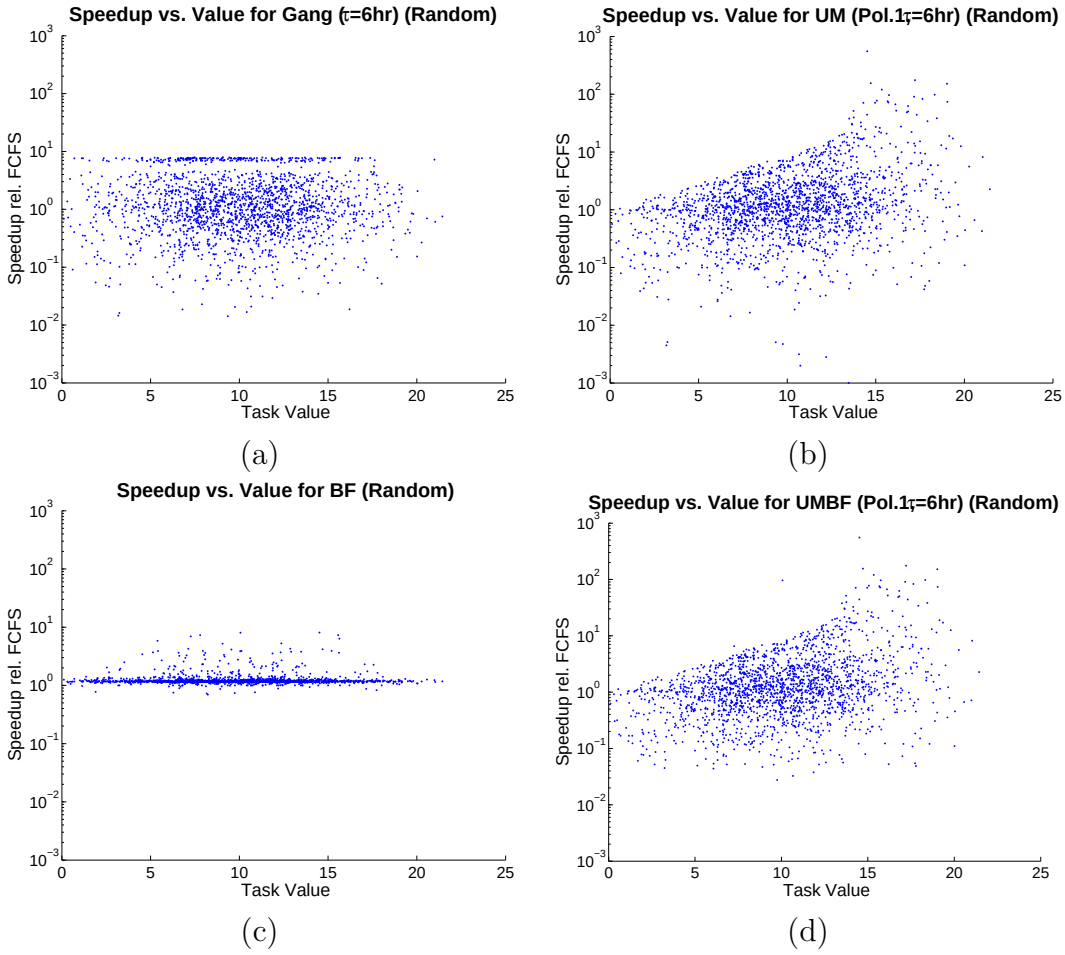

Figure 4.3: Task speedup vs. value for the random workload using (a) Gang, (b) UM policy 1, (c) BF, and (d) UM+BF policy 1.

tasks are experiencing larger speedups as compared to the lower valued ones.

A final insight can be obtained by observing figure 4.4, a scatter plot of the task startup delays versus credit-value. It is clear that for the Gang and BF strategies, there is no relationship between credit-value and startup delay. Notably, however, the time-slicing nature of Gang results in relatively short startup delay; however, the overall completion time was previously shown to not benefit from this early start. In contrast, UM policy 1 (figure 4.4 (b)) clearly shows that higher valued tasks have lower startup delays. UM+BF also shows this relationship, however a number of exceptional tasks break the pattern seen in 4.4 (b) – these tasks benefit from the idle resources by being backfilled without regard to their credit-value.

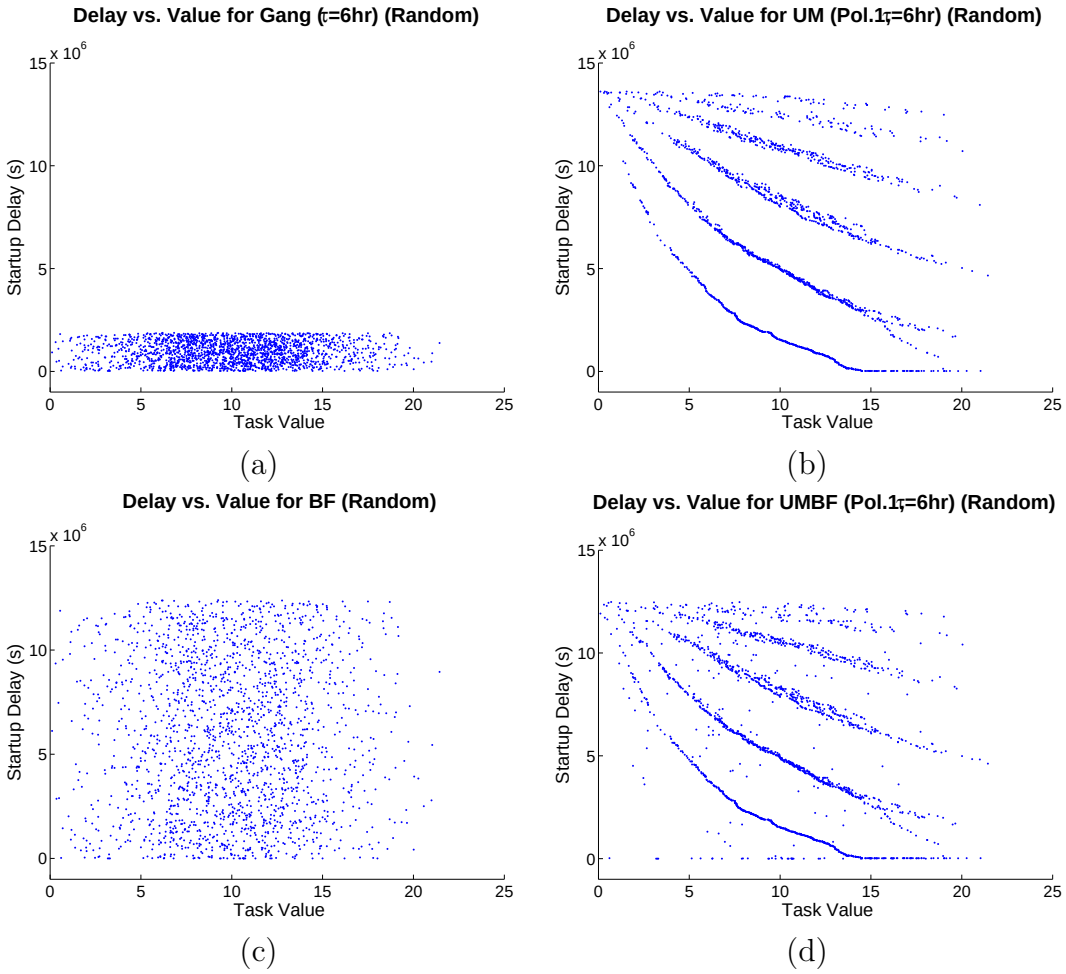

Figure 4.4: Task startup delay vs. value for the random workload using (a) Gang, (b) UM policy 1, (c) BF, and (d) UM+BF policy 1.

The banding observed in the UM and UM+BF plots is an interesting result; it is caused by the scheduler's preference for a high value/resource cost ratio (i.e. single processor tasks are preferred over dual processor tasks when both have the same assigned credit-value). This result is made clear in the next section, when we observe a similar pattern in the NPACI workload results.

4.5.2 Experimentation with the NPACI Workload

In section 4.5.1, we demonstrated that under a randomly-generated workload, the UM and UM+BF allocation strategies can improve job throughput and that the incorporation of a credit-value can allow users to intervene with urgent tasks without

negatively affecting the overall schedule characteristics. In this section, we evaluate the strategies using real workloads. The significance of this additional evaluation results from the fact that real workloads commonly feature irregularities that can negatively affect the schedulers performance.

The selected workload for this study is derived from traces obtained from the NPACI JOBLOG Job Trace Repository [63]. The JOBLOG provides job traces of a 128 node IBM SP system at San Diego Supercomputing Center and includes two years of data from May 1, 1998 until April 30, 2000 [63]. Each task is described by metadata defining, among other characteristics, its submission, start and end times, and the number of requested and allocated processors.

Figure 4.2 (right) shows a histogram of the task lengths found in the JOBLOG. The JOBLOG features a large proportion of short tasks (less than 20000s), with a long tail of larger tasks reaching 5.7×10^6s at the far right. The mean interarrival time is 1114s. Tasks which were cancelled early and before completing successfully are excluded from this study.

Experimental Setup

To evaluate the results of the NPACI workload, we simulated a grid consisting of four clusters with 16, 16, 32, and 128 homogeneous processors each. Other grid sizes are investigated in section 4.5.3. Similar to the random workload, we investigate various workloads by scaling the interarrival time λ by k_λ, which is varied from 0 to 1, where 0 results in a very heavy workload, and 1 results in a light workload.

The NPACI workload is derived by extracting the arrival time, execution time, and number of processors from the JOBLOG. Because simulation of the entire JOBLOG is too computationally intensive, we extract a subset of tasks starting at a random offset in the JOBLOG and continuing for 5000 tasks. We present the average results of 10 of these subset workloads. Since the JOBLOG does not specify malleability options for each task, we have randomly generated these options by varying the

Table 4.3: Overall completion time (units of 10^7s) for the NPACI workload and $\tau = 6hr$

Strategy	$k_\lambda = 0$	$k_\lambda = 0.5$	$k_\lambda = 1.0$
FCFS	1.94 ± 0.47	2.29 ± 0.49	4.22 ± 0.83
BF	1.70 ± 0.41	2.21 ± 0.45	4.22 ± 0.83
m-m	1.74 ± 0.41	2.24 ± 0.47	4.22 ± 0.83
Gang	5.62 ± 1.07	5.49 ± 1.03	5.41 ± 0.99
UM Pol.1	5.41 ± 1.02	5.35 ± 1.01	5.43 ± 1.02
UM Pol.2	5.60 ± 1.08	5.55 ± 1.07	5.37 ± 1.00
UM Pol.3	4.45 ± 0.85	4.65 ± 0.89	4.76 ± 0.87
UM Pol.4	4.11 ± 0.82	4.13 ± 0.81	4.56 ± 0.83
UM Pol.5	4.04 ± 0.79	4.08 ± 0.80	4.53 ± 0.83
UM+BF Pol.1	1.71 ± 0.43	2.23 ± 0.46	4.22 ± 0.83
UM+BF Pol.2	1.70 ± 0.42	2.21 ± 0.45	4.22 ± 0.83
UM+BF Pol.3	1.73 ± 0.44	2.21 ± 0.45	4.22 ± 0.83
UM+BF Pol.4	1.71 ± 0.43	2.21 ± 0.45	4.22 ± 0.83
UM+BF Pol.5	1.72 ± 0.43	2.22 ± 0.45	4.22 ± 0.83

number of requested processors and correspondingly scaling the running time (e.g. a 64 processor task may have options corresponding to 64, 32, and 16 processors with running times of 1, 2, and 4 hours, respectively). The credit-value assigned to each task is assumed to be normally distributed using the same parameters as for the random workload (i.e. the mean of the maximum values is 10 and the standard deviation is 4).

Results of the NPACI Workload

For each of the studied strategies and policies, we have obtained a set of results for comparison to those from the random workload. Tables 4.3 and 4.4, as well as figures 4.5 and 4.6 present the simulation results. Note that the tables highlight the important results, and complete data tables are available in appendix 1.

In table 4.3, the overall completion time for each allocation strategy is presented. Under all loading levels, all of the UM+BF policies perform as well as the best reference strategies. The results are the same for all of the UM+BF policies, implying that the BF component of UM+BF is contributing most to the strategy's high efficiency. Indeed, the pure-UM policies perform significantly worse than UM+BF under

Table 4.4: Correlation of credit-value with speedup for the NPACI workload and $\tau = 6hr$.

Strategy	$k_\lambda = 0$	$k_\lambda = 0.1$	$k_\lambda = 0.2$
BF	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01
m-m	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01
Gang	-0.01 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01
UM Pol.1	0.39 ± 0.04	0.13 ± 0.05	0.10 ± 0.03
UM Pol.2	0.17 ± 0.04	0.13 ± 0.03	0.08 ± 0.01
UM Pol.3	0.18 ± 0.03	0.13 ± 0.03	0.07 ± 0.01
UM Pol.4	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01
UM Pol.5	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01
UM+BF Pol.1	0.10 ± 0.04	0.00 ± 0.01	0.00 ± 0.01
UM+BF Pol.2	0.04 ± 0.01	0.00 ± 0.01	0.01 ± 0.01
UM+BF Pol.3	0.05 ± 0.02	0.00 ± 0.01	0.00 ± 0.01
UM+BF Pol.4	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
UM+BF Pol.5	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01

heavy and moderate loading. Pure-UM's performance is marginally improved by the incorporation of time-based factors in policies 4 and 5. It is evident that the NPACI workloads present too many small tasks for the pure-UM strategy to produce timely schedules. In order to improve the efficiency of the schedules, it is clear that more frequent reallocations are necessary.

The correlation of credit-value with per-task speedup relative FCFS is shown in table 4.4. Once again, the largest correlations are seen with pure-UM policies 1, 2 and 3; however, the cost of this performance is in sacrificed efficiency, as shown in the previous table. When backfilling is employed, the correlation drops significantly (and disappears under moderate or light loading).

The effect of the credit-value on the resulting schedules is depicted further in figures 4.5 and 4.6. The speedup versus credit-value is shown in figure 4.5: we see that UM policy 1 results in the largest speedups for higher valued tasks. This is expected because the policy 1 uses credit-value exclusively in its utility function, whereas the other policies use time-based metrics to sacrifice this correlation for better overall completion times. The reference strategies, as expected, result in speedups which are not related to value. Additionally, UM+BF has most of its tasks scheduled by the

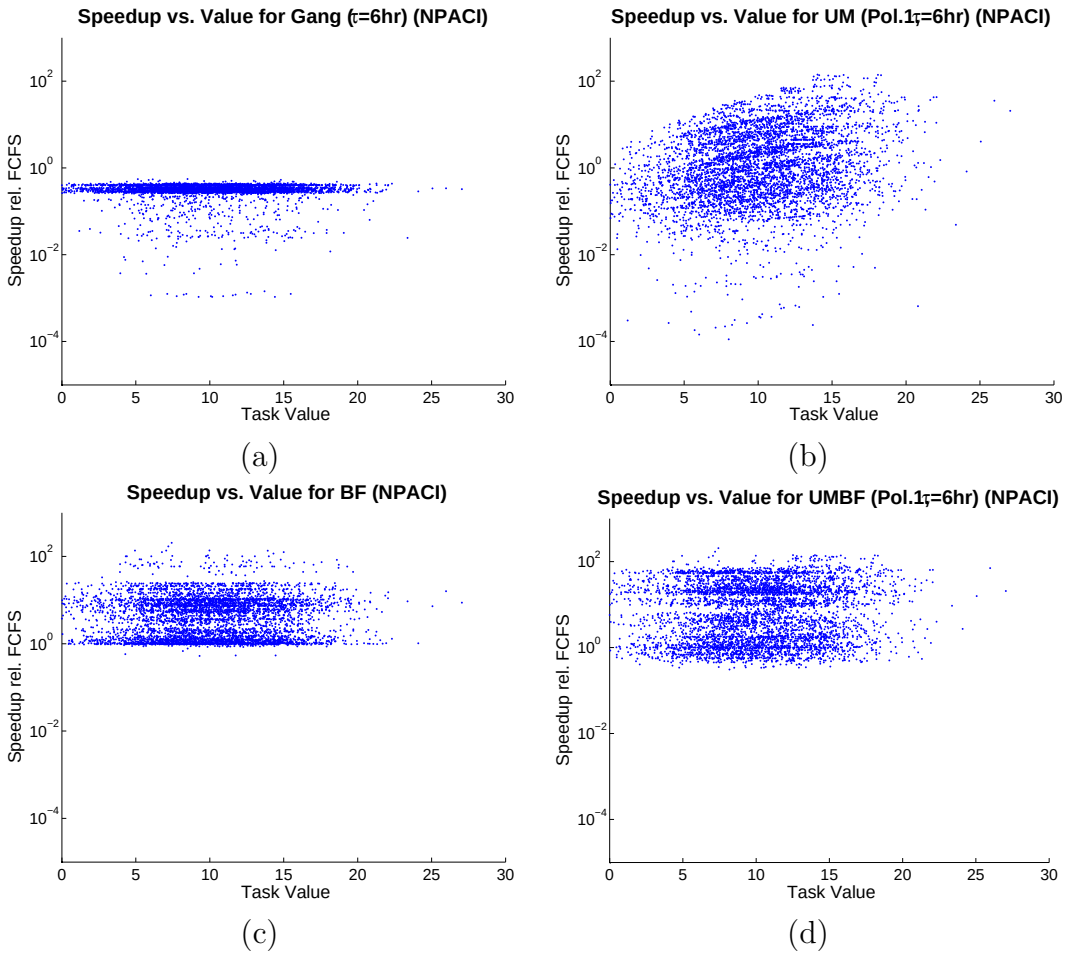

Figure 4.5: Task speedup vs. value for the NPACI workload using (a) Gang, (b) UM policy 1, (c) BF, and (d) UM+BF policy 1.

backfilling strategy; it is therefore unable provide the positive correlation between speedup and credit-value.

In figure 4.6 we observe the startup delays for four strategies. UM policy 1 shows how higher valued tasks are associated with shorter delays. Additionally, the recognizable banding effect is again apparent in the UM policy. By scaling the y-axis of this plot, we can clearly observe the cause. Figure 4.7 depicts the delay vs. value plot for the earliest tasks scheduled using UM policy 1. In this plot, we can observe two bands; for a given delay, the right band appears at a value roughly twice that of the left band. The banding is therefore explained as follows. During the first allocation cycle, the highest value tasks requiring only one processor are

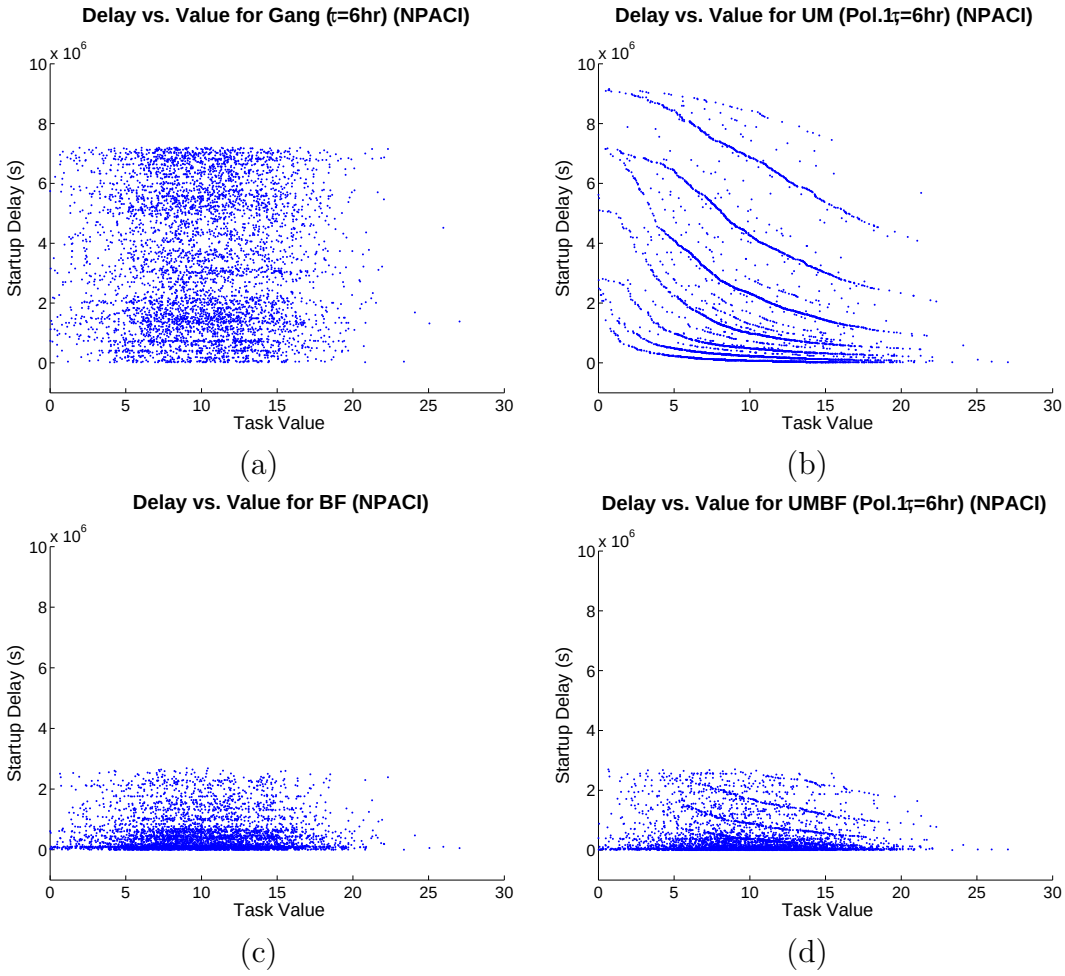

Figure 4.6: Task startup delay vs. value for the NPACI workload using (a) Gang, (b) UM policy 1, (c) BF, and (d) UM+BF policy 1.

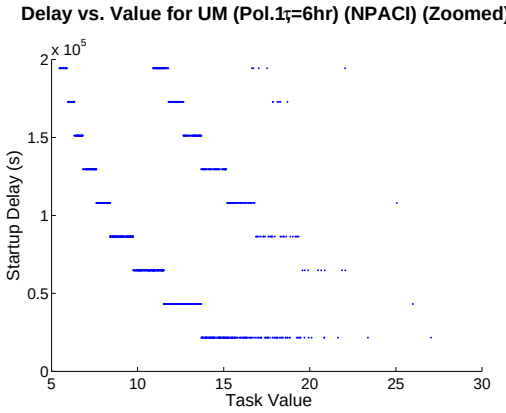

Figure 4.7: Task startup delay vs. value for the NPACI workload and UM policy 1 zoomed to show the startup of early tasks.

selected. These tasks are associated with the highest value per unit resource in the entire workload. Two-process tasks are therefore selected only when their values are greater than the two highest valued single-processor tasks. Tasks requesting larger numbers of processors follow this behaviour, resulting in the banding. Note that figure 4.7 also demonstrates that startup delays are quantized to multiples of 6 hours; this corresponds to the allocation cycle time τ and is the result of not allowing backfilling between allocations.

It is clear from the NPACI workload results that a shorter reallocation period is required for scheduling a large number of small tasks using the utility/knapsack approach. Indeed, more frequent reallocations are useful to solve two sources of suboptimal performance: (1) resources become available yet are not utilized until the next reallocation cycle; and (2) a highly-valued task arrives in the queue, and should preempt a lower-valued running task, yet the preemption does not occur until the next reallocation cycle. Backfilling is effective in solving (1) when there are short tasks in the queue; however, these tasks are not serviced in manner respecting the quality-of-service metrics so the optimality of the allocation is not maintained. By employing more frequent reallocations, both of these problems can be minimized. Indeed, one should use the shortest reallocation period possible, though it would generally not be beneficial to reallocate more frequently than the rate at which tasks arrive or the rate at which resources become available. In a closed grid system, where the computational demand matches the grid's capabilities, these two rates are roughly equivalent; therefore, reallocating at this rate is optimal.

Note that, in practise, frequent reallocations must be weighed against the checkpointing and reallocation overheads. One method to incorporate these overheads is presented in chapter 6, where the data transmission overhead related to migrating tasks is considered in a utility heuristic.

A subject of future research could be to evaluate a utility/knapsack approach with

Table 4.5: Comparison between Overall Completion Times and Delay-Value Correlation for Small and Large Grids

	Pol. 1	Pol. 2	Pol. 3	Pol. 4	Pol. 5
Overall Completion Time ($*10^6$ s)					
Small	15 ± 3	16 ± 3	15 ± 3	14 ± 3	15 ± 3
Large	2.3 ± 0.4	2.3 ± 0.4	1.9 ± 0.3	1.6 ± 0.3	1.5 ± 0.3
Correlation of Delay w/ Max. Value					
Small	-0.970 ± 0.005	-0.21 ± 0.03	-0.20 ± 0.04	0.005 ± 0.006	0.005 ± 0.006
Large	-0.980 ± 0.001	-0.21 ± 0.03	-0.20 ± 0.03	0.005 ± 0.005	0.005 ± 0.006

aperiodic reallocations. In such a system, reallocations would be performed whenever a new task arrives, or when a resource becomes available. This would ensure that the optimality of the task placements is always maintained.

4.5.3 Evaluation on Varied Grid Sizes

In the above studies, we investigated the effects of varied workloads by scaling the task interarrival times for the different experiments. Whereas this technique presents results for light and heavily loaded grids, it does not directly indicate results for scheduling tasks on grids with fewer or greater numbers of processors. In this section, we investigate the performance of the utility/knapsack approach for scheduling NPACI workloads on grids which are smaller and larger than the grid assumed in section 4.5.2. For the *small* grid, we assume a grid of four clusters having 4, 8, 16, and 32 processors, respectively; note that this is the same size as the grid used in the random workload study. The *large* grid corresponds to a grid of four clusters having 32, 64, 128, and 256 processors, respectively; this grid has 8 times more processors than the small grid. The NPACI workloads used for this evaluation are identical to those in section 4.5.2 having $k_\lambda = 0$ and the allocation period is assumed to be $\tau = 6$ hrs.

The results are presented in table 4.5. For the small grid, we observe that the overall completion time is independent of the policy used. This is because the small grid limits the number of options for each task; for example, a 32 process task has only

one placement option. In this restricted environment, the policies generally result in the same placements, which leads to similar overall completion times. For the large grid, we observe that the overall completion time is dependent on the policy used. In general, the credit-value policies (1, 2, and 3) are slower than the time-based policies (4 and 5). The large grid, having a greater number of placement options for each task, provides better optimization potential for the time-based utility heuristics.

In the lower half of table 4.5, we can review the correlation between delay and value; the ideal correlation is -1, implying that the most highly valued tasks are delayed the shortest. The results of both the small and large grids are generally equivalent, with policy 1 producing a nearly optimal correlation. This similarity between the small and large grids allows us to conclude that the utility/knapsack approach can preferentially allocate highly valued tasks on varied grid sizes.

4.5.4 Direct Comparison of Random and NPACI Workloads

The previous sections presented results for random and NPACI workloads under varied workload and scheduler configurations. The evidence has shown that the long tails and high proportion of short tasks which characterize the NPACI workloads lead to decreased utilization of the grid resources. In this section, we complete this argument by extracting NPACI workloads that are similar in computational demand to the random workloads and comparing the results when all scheduling parameters are equal. We simulate a grid of 4, 8, 16, and 32 processors, and the scheduler is configured to use the Policy 1 utility heuristic at an allocation frequency of $\tau = 6$ hours. In all cases, the entire task workload is assumed to have arrived at time zero ($\lambda = k_\lambda = 0$).

The results of this experiment are given in table 4.6. The table presents the mean results of the random workloads used in section 4.5.1. We have also identified two NPACI-derived workloads, denoted *NPACI-4* and *NPACI-1*, which have computational demands that are similar to the random workloads. The table reports four

Table 4.6: Comparison of the Average Random Workload Results and the NPACI-derived Workload Results

	Random		NPACI-4		NPACI-1	
	UM	UM+BF	UM	UM+BF	UM	UM+BF
Computational Demand (units of $*10^8$ CPU-secs)	7.9	7.9	8.5	8.5	5.2	5.2
Overall Completion Time (units of $*10^7$ secs)	1.4	1.3	2.8	1.8	1.8	1.0
Efficiency	0.91	0.98	0.51	0.80	0.80	0.90
Value-Delay Correlation	-0.42	-0.38	-0.94	-0.33	-0.33	-0.26

metrics: computational demand, measured in CPU-seconds; overall completion time, measured in seconds; efficiency, computed as the computational demand divided by the product of overall completion time and grid size; and the correlation of credit-value and task startup delay, used to indicate preferential treatment of high valued tasks.

First, we inspect the results of NPACI-4, which differs from the random workloads in computational demand by roughly 7%. For NPACI-4, though the demand is similar to the random workloads, the resulting overall completion times vary greatly. When scheduling the random workloads using the UM strategy, the scheduler is able to complete all of the tasks within $1.4 * 10^7$ seconds, leading to a 91% efficiency; this is increased to 98% when UM+BF is used. On the other hand, NPACI-4 requires almost twice as long to complete, leading to an efficiency of 51%. When UM+BF is used for NPACI-4, the efficiency is increased to 80%.

The correlation results show that both UM and UM+BF strategies preferentially allocate high-valued tasks for both the random workloads and NPACI-4. Indeed, NPACI-4 results in nearly optimal correlation at -0.94. The introduction of backfilling leads to compromised performance for both workloads, which is expected because credit-value is not considered in the backfilling procedure.

The second NPACI-derived workload is NPACI-1, which has computational demand of roughly 30% less than that of the random workloads. The distribution of

tasks in NPACI-1 allows it to be scheduled more efficiently than NPACI-4, though it is still less efficient than the random workloads. Also, while the NPACI-1 correlation results do not match those of NPACI-1, they do indicate that highly valued tasks were still allocated preferentially. The differences in the results between NPACI-4 and NPACI-1 reflect the large variations over time in the NPACI workload.

We conclude that the distribution of task lengths in NPACI-4 and NPACI-1, featuring the same general characteristics as the entire NPACI JOBLOG, lead to inefficient resource utilization under these scheduling parameters. However, the negative correlations between credit-value and startup delay indicate that high-valued tasks are preferentially allocated regardless of the workload.

4.5.5 Evaluation on a Heterogeneous Grid

In the random and NPACI workload studies above, the grid resources were assumed to be homogeneous; that is, all of the processors on the grid were equivalent in capability and performance. In this section, we evaluate the performance of the utility/knapsack approach on a heterogeneous grid. In particular, we have simulated the pure UM strategy using the five policies introduced in section 4.4; however, in this case, the performance of each grid cluster is a uniform random value between 0.5 and 1.5 times the reference processor. For each of 100 experiments, 500 tasks drawn from a random offset in the NPACI JOBLOG are submitted at time zero to the grid and the pure UM strategy is used to perform the allocations. For a fair comparison between the homogeneous and heterogeneous grids, the performance of each grid is normalized to each other. The normalization condition employed here is

$$\sum_{k=1}^{m} P_{hetero_k} R_k = \sum_{k=1}^{m} P_{homo_k} R_k \tag{4.13}$$

where P_{hetero_k} and P_{homo_k} are the cluster performance values for the heterogeneous and homogeneous grids, respectively, and R_k is the number of processors at cluster

Table 4.7: Comparison between Overall Completion Times and Delay-Value Correlation for Homogeneous and Heterogeneous Grids

	Pol. 1	Pol. 2	Pol. 3	Pol. 4	Pol. 5
Overall Completion Time ($*10^5$ s)					
Hom.	6.6 ± 0.4	6.4 ± 0.4	5.4 ± 0.3	5.0 ± 0.3	5.0 ± 0.3
Het.	6.5 ± 0.4	6.4 ± 0.4	5.4 ± 0.3	5.0 ± 0.3	4.9 ± 0.3
Correlation of Delay w/ Max. Value					
Hom.	-0.962 ± 0.002	-0.25 ± 0.02	-0.25 ± 0.02	0.003 ± 0.005	0.004 ± 0.005
Het.	-0.962 ± 0.002	-0.25 ± 0.02	-0.25 ± 0.02	0.003 ± 0.005	0.003 ± 0.005

k.

The results are presented in table 4.7, with the overall completion time shown in the upper portion of the table, and the correlation between task startup delay with the value assigned to a task in the lower half of the table. The running time results show the typical performance results that were observed previously: the incorporation of the estimated response time and nearness to completion time metrics improve the efficiency of the schedule. In addition, the pure credit value heuristic in policy 1 achieves the largest correlation between value and startup delay; the other heuristics sacrifice this prioritization for overall schedule efficiency.

When comparing the results between the homogeneous and heterogeneous cases, it is apparent that the heterogeneous grid has slightly faster overall completion time for policies 1 and 5; however, there is no significant difference in the performance between the two grids. The utility/knapsack approach is therefore shown to be applicable to a wider range of grids.

4.5.6 Knapsack Solution Time and Scalability

An allocation of a few tasks to a small number of grid resources is a simple problem that can be solved using a brute-force method in a short amount of time. However, in cases where the number of tasks and grid resources is large, the problem becomes difficult to solve within a reasonable amount of time. It is therefore useful to observe the time required to solve the allocation problems presented in this chapter.

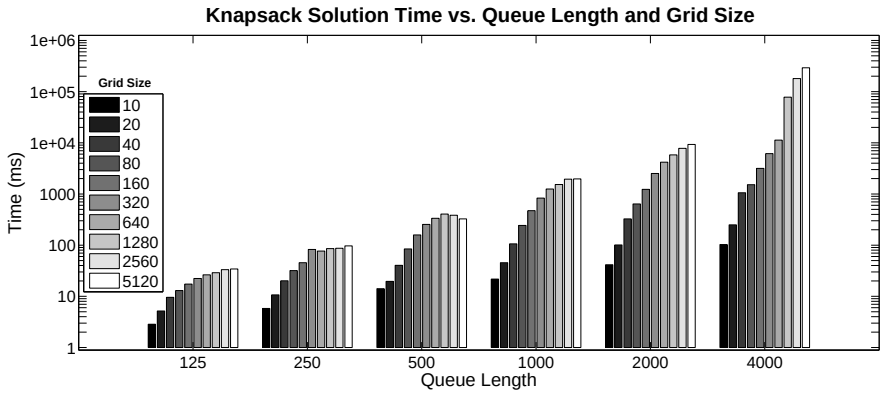

Figure 4.8: Knapsack problem solution time for various queue lengths and grid sizes.

Figure 4.8 presents the knapsack solution time for a variety of task queue lengths on a number of grid sizes. The times were obtained by solving the knapsack problem on a dual core Opteron 275 running at 2.2 gigahertz with 8 gigabytes of memory. For queue lengths of less than 1000 tasks, the knapsack problem is solved in under 1000ms. For larger queue lengths, the solution time depends highly on the grid size. The longest solution time, corresponding to the allocation of 4000 tasks to a grid of 5120 processors, is roughly 200 seconds. This shows that even with very large problems, nearly optimal solutions can be obtained in a short amount of time relative to the reallocation periods.

Further, the time taken to solve the allocation problem can be hidden by parallelizing the knapsack solver with the running tasks. This technique starts optimizing the allocation problem a number of seconds prior to the checkpoint time. This way, the new allocation is available at checkpoint time without delay. Though the state of the grid is not identical to that at checkpoint time, it is unlikely to change greatly during the solution time.

4.5.7 Experimental Conclusions

In this section we evaluated the utility/knapsack strategies using two workloads, one which was randomly generated and one which is drawn from a real system. Using the

random workload, we demonstrated that the UM strategy is able to produce efficient schedules and that the assignment of a credit-value allows users to intervene with urgent tasks. Using the real workload, we observed that the long-tailed distribution of task lengths combined with the very large ratio of short tasks resulted in less optimal schedules; for the strategies without backfilling, resources were idle for too long and the resulting efficiency was poor. While the incorporation of backfilling improved efficiency, it significantly decreased the ability of the utility-based scheduler to implement its allocation policies. It is clear that more frequent reallocations will improve the efficiency of the UM strategy; however, it is unclear what overhead cost this will incur.

It is important to note that while the UM and UM+BF strategies had problems with the NPACI workload at these allocation periods, it will not necessarily perform poorly on all real workloads. For real workloads featuring a smaller proportion of short tasks, it is expected that the utility-based strategies will perform more like they did for the random workload. Indeed, it can be concluded that due to the overhead involved with checkpointing and reallocating, the utility-based strategies are ideally suited to environments rich in long tasks.

4.6 Conclusions

In this chapter we have introduced a utility model for the allocation of grid resources which, when solved using a formulation as a variant of the 0-1 multichoice multi-dimensional knapsack problem, finds allocation sets which are congruent with the desired allocation policies. The solution presented employs Quality of Service concepts in the grid resource allocation problem, and shows that QoS-enabled policies do indeed allocate resources efficiently and allow for preferential allocation to highly valued tasks with no detriment to the overall completion time of the submitted tasks as a group. Further, the notion of utilizing external credit-value metrics ensures

both that users will be able to differentiate and intervene based on the urgency of a particular task, and establishes an economic base to the grid.

The evaluation of the utility-based strategies was performed in simulation against both randomly-generated and real task workloads. The credit-value-enabled policies were able to schedule tasks preferentially based on their assigned value while keeping the overall system utilization to the maximum. The long-tailed properties of the NPACI workload have presented interesting challenges and the performance of the allocation policies need to be further explored under such non-ideal but real conditions.

Chapter 5

Design and Analysis of the Utility/Knapsack Scheduling System

This chapter employs two commonly used design and analysis techniques to achieve a better understanding of the utility heuristics presented in chapter 4. Using conventional sensitivity analyses, the weights of complementary utility metrics are varied in order to discover their effects on resulting allocations. Next, the Plackett-Burman design methodology is used to further understand the parameters in a single utility function, leading to further relationships which were not apparent in the sensitivity analyses. These two techniques are shown to be effective in understanding the system parameters, leading to a methodology for optimizing the utility/knapsack technique when it is used in practise.

The work presented in this chapter was published in the Proceedings of ICS'06, the 20th Annual International Conference on Supercomputing [66], and the Proceedings of HPCS'07, the 21st International Symposium on High Performance Computing Systems and Applications [67].

5.1 Introduction

In the preceding chapter, this dissertation introduced a novel resource allocation strategy for computational grids based on a utility model and solved it using a for-

mulation as a knapsack problem. The objective of the scheduling system is to perform these resource allocations in such a way that grid resources are utilized efficiently, jobs are scheduled promptly, and all users are treated fairly. When grid scheduling systems are used in practise, their design is typically quite complex, involving a large number of configuration parameters whose effects and interactions are not well understood. This complexity often results in designs than are *ad hoc*, suboptimal, and difficult to maintain. This chapter presents two design and analysis techniques which can be used to help designers to better understand the design parameters, thereby helping them achieve a well designed scheduling system.

The first technique shown in this chapter is to perform sensitivity analyses on utility function heuristics introduced in chapter 4. In the previous chapter, the utility functions explored included terms for various QoS metrics, including credit-value, estimated response time, and nearness to completion time; however, their relative weights were chosen to be equal, which is not necessarily the ideal configuration. By performing sensitivity analyses, we can achieve two results. First, the analyses allows for verification that the metrics are performing as intended; as the weight of a given metric is increased, its intended result should become more clear. Second, by varying the relative weights of these metrics the optimal value or range of values can be observed.

The second technique presented in this chapter is to apply the Plackett-Burman design methodology [68, 69] to the selection of parameters used by the utility/knapsack system. Plackett-Burman is a design technique which is used to find the effects of design parameters while requiring a limited number of experiments. In this work, it is shown that this design methodology can be used to achieve a better understanding of the utility/knapsack configuration parameters, thereby leading to better performance.

The remainder of this chapter is organized into two main sections, namely, sec-

tion 5.2 which presents the sensitivity analyses, and section 5.3 which presents the Plackett-Burman design methodology and results. Section 5.4 concludes.

5.2 Sensitivity of the Allocation Policies

In this section, we seek to understand the sensitivity of UM-derived allocations to variations in the resource allocation policies. These policies describe the relative preference of each task submitted to the grid and are evaluated to find a utility value for each given task option. For each task option, we compute a QoS-metric-parameter P_{ij} using equation 4.1. The variation of the weights w_l in the QoS-metric-parameter P_{ij} allows different policies to be implemented. In this study, we select combinations of complementary QoS metrics and characterize the schedules resulting from varied relative weights of these metrics.

5.2.1 The External Credit-Value Metric

Our first analysis seeks to observe the relationship between resulting task schedules and the incorporation of an external user-assigned credit-value metric. In this experiment, the task option utility is computed using

$$u_{ij} = w_v \frac{v_{ij}}{v_{max}} + (1 - w_v) S \left(\frac{t - s_i + r_{ij}}{N_i} + \frac{N_i}{r_{ij}} \right) \tag{5.1}$$

$$S(x) = \frac{2}{e^{-5x/x_{max}} + 1} - 1 \tag{5.2}$$

where w_v is the weight of the value metric (normalized to v_{max}, the largest credit-value observed in the current allocation cycle). The right-hand-side (RHS) of the utility equation prefers tasks according the intrinsic estimated response time and nearness to completion time metrics, where t is the current time, s_i is the submitted time, r_{ij} is the remaining time under option j, and N_i is the total task length. Note that the entire RHS is normalized using equation 5.2, where x_{max} is the largest value of x observed in the current allocation cycle. This normalization ensures that the

domain of the sigmoidal normalizing function S is $[0:1]$.

5.2.2 The Intrinsic NTCT Metric

The second analysis is designed to observe the effects of varying the weight of the nearness-to-completion-time (NTCT) metric relative to the estimated response time metric. NTCT is included to prefer tasks that are close to finishing, rather than postponing their completion until resources become less busy. We compute the task option utility using the metrics defined above and mediate the response time with NTCT weight w_n:

$$u_{ij} = w_n \frac{N_i}{r_{ij}} + (1 - w_n) \frac{(t - s_i + r_{ij})}{N_i} \tag{5.3}$$

5.2.3 Shaping the Sigmoidal Utility Function

In our final analysis we study the effect of varying the shape of the utility function $U(\cdot)$ on resulting task schedules. The QoS metrics selected are the estimated response time mediated by the NTCT. Each metric is given an equal weight in this experiment. The utility function used in this analysis is

$$u_{ij} = U\left(\frac{t - s_i + r_{ij}}{N_i} + \frac{N_i}{r_{ij}}\right) \tag{5.4}$$

where the QoS-metric-parameter is normalized using the utility function

$$U(x) = \frac{2}{e^{-\alpha x / x_{max}} + 1} - 1 \tag{5.5}$$

By varying α, the shape of the sigmoidal curve is affected. Examples of the dependence on α are shown in figure 4.1. In this figure we observe that small values of α result in almost linear functions of small values, whereas large values of α produce saturated curves.

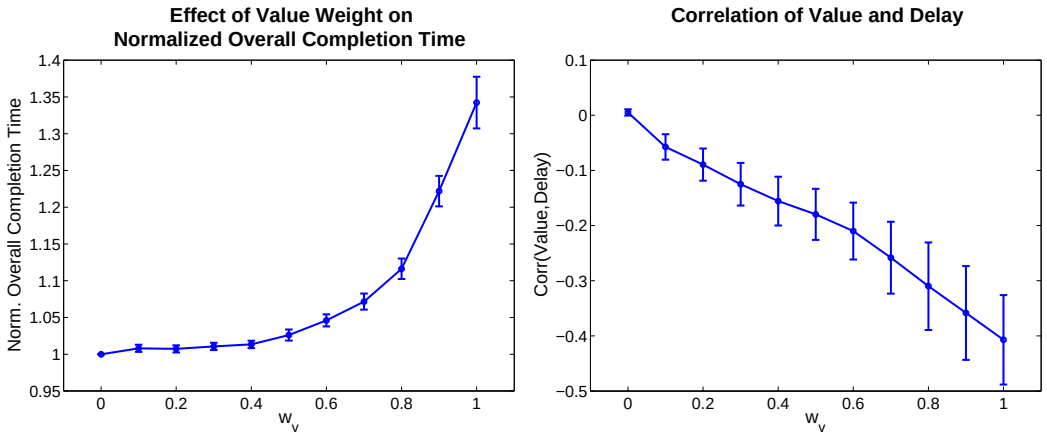

Figure 5.1: Effect of the value weight w_v on the normalized overall completion time (left) and the correlation between value and startup delay (right).

5.2.4 Experimentation and Results

The experimental configuration and task workloads for the sensitivity analyses is identical to that of the real workload simulations presented in section 4.5.2. However, we limit the analyses to the case when $k_\lambda = 0$ and $\tau = 6$hrs.

Simulation results are given in figures 5.1 through 5.4. Figures 5.1, 5.3, and 5.4 show errors of σ_μ above and below the mean μ.

Results of the Credit-Value Sensitivity Analysis

To measure the effect of varying the credit-value metric we simulate schedules using the task utilities given in equation 5.1. Varied relative strengths of the credit-value metric are effected by varying w_v from 0 to 1 in increments of 0.1.

Results of this section are presented in figures 5.1 and 5.2. Figure 5.1 (left) depicts the effect of increasing w_v on the overall completion time. This figure shows normalized time values; results are normalized to the $w_v = 0$ case for each random seed, then mean results for all seeds are presented. At $w_v = 0$, the absence of any credit-value contribution to the utility results in the most efficient schedules; at this point, equation 5.1 is computed using only intrinsic time-based metrics. At $w_v = 1$, the overall completion time is roughly 35% longer – it is clear that pure credit-value

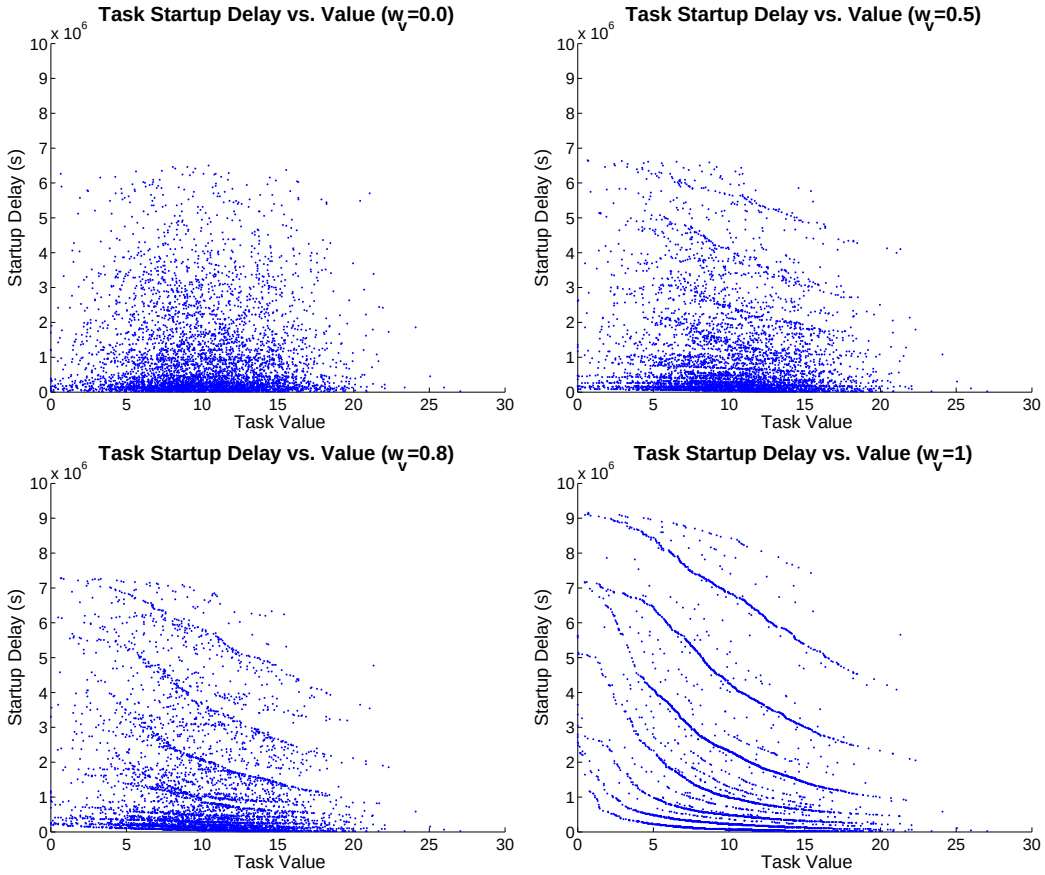

Figure 5.2: Scatter plots of the task startup delay versus task value for various credit-value weights.

metric is costly in terms of resource utilization. However, as w_v is decreased from 1, the efficiency is quickly improved, and at $w_v = 0.5$, the cost is less than 3%.

Because the goal of the value metric is to effectively raise the priority of highly valued tasks, the effectiveness of the metric on the schedule can be determined by measuring the correlation of the credit-value and the startup delay for each task. A strong negative implies that higher valued tasks are receiving preferential treatment from the scheduler. In figure 5.1 (right) we present this correlation. By increasing w_v from 0 to 1, the correlation is shown to decrease linearly from 0 to -0.4. Previously, the efficiency was shown to be degraded at roughly $w_v \geq 0.8$. These results show that a correlation of -0.25 is achievable with a 5% cost of efficiency.

A clear picture of the w_v's effect on startup delay is shown in figure 5.2. In these

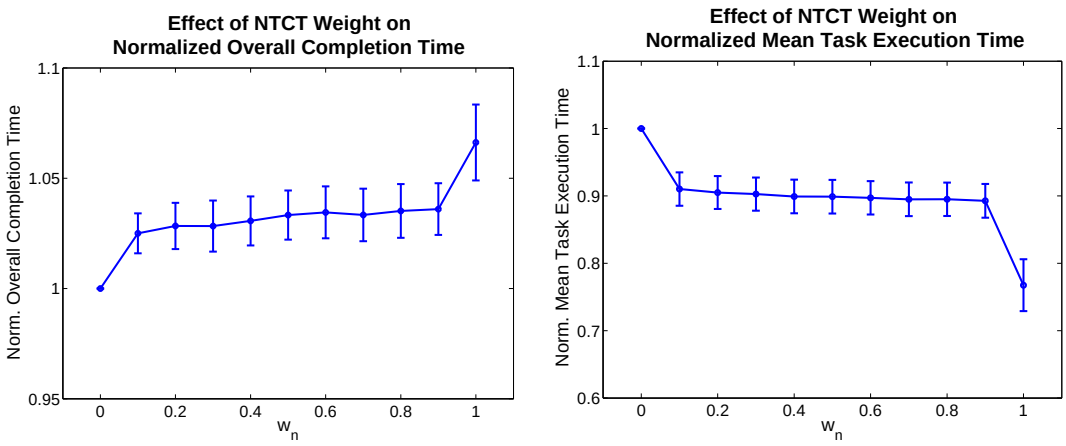

Figure 5.3: Effect of the NTCT weight w_n on the normalized overall completion time (left) and normalized mean task execution time (right).

scatter plots, we present the startup delay versus credit-value when w_v is 0, 0.5, 0.8, and 1. When $w_v = 0$, we see no correlation between value and delay (the clustering around a value of 10 reflects the Gaussian distribution around this point). At the other extreme, when $w_v = 1$, we see the familiar plot showing the preference for high-value tasks and the pattern of processor number bands. For $w_v = 0.5$ and 0.8, this banding becomes less clear, as the time-based metrics begin preferring tasks other than those with large credit-values.

Results of the NTCT Sensitivity Analysis

The effect of mediating the response time metric with the nearness to completion time (NTCT) metric is demonstrated in figure 5.3. These results are obtained by simulating task schedules using the utility given in equation 5.3.

Figure 5.3 (left) presents the effect of varying w_n on the normalized overall completion time. Interestingly, this workload is most efficiently scheduled when $w_n = 0$, corresponding to a pure ERT policy. The introduction of the NTCT metric increases the overall completion time around 2.5%. It is therefore concluded that the incorporation of the NTCT metric decreases resource utilization. However, the benefit of NTCT is not efficiency, but instead it is introduced to prevent tasks that are nearly

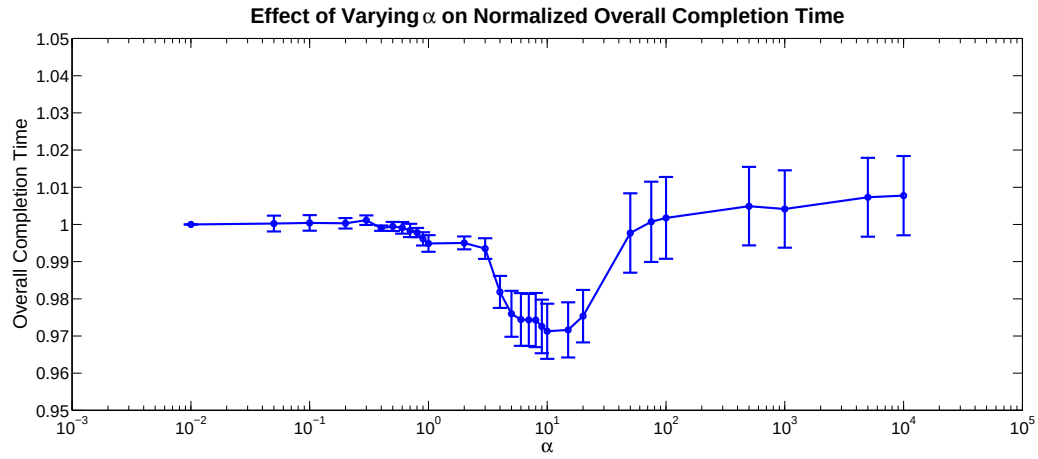

Figure 5.4: Effect of varying the shape of the sigmoidal utility function on the normalized overall completion time.

completed from being preempted (late preemptions). This effect can be observed by measuring the mean execution time (completion time minus start time) of each task in the workload. Tasks that are preempted often will see an increase in their execution time, so since the NTCT metric prevents late preemptions, the mean execution times should decrease. As expected, figure 5.3 (right) shows this relationship. Even though the introduction of NTCT increases the overall completion time, the mean execution time of each task is decreased. Based on these results, it can be concluded that $w_n = 0.1$ effectively prevents late preemptions without a large decrease in efficiency.

Results of the Sigmoidal Sensitivity Analysis

In our final analysis we seek to determine the effect of varying α in the sigmoidal utility function given in equation 5.5. In this experiment we simulate task schedules using the task option utility given in equation 5.4.

Figure 5.4 presents the effect of varying α on the overall completion time. The overall completion times are each normalized to the value at $\alpha = 10^{-2}$. As depicted earlier in figure 4.1, low values of α result in an effectively linear function with small normalized values, whereas high values of α result in an early saturation of

the function at low values of x. In figure 5.4, we see that utility functions having $\alpha \leq 1.2$ result in largely similar completion times. This is expected because the relative linearity of the function maintains the relative value of the tasks. For utility functions having non-saturating, yet non-linear, values of α ($1.2 < \alpha < 20$), we see a decrease in the completion time, with a peak decrease at $\alpha = 10$. In the utility functions which saturate ($\alpha > 20$), we see an decrease in the scheduling efficiency. In these cases, tasks having values at or above the saturation point will receive equal preference, and therefore the schedule will behave unpredictably.

5.2.5 Discussion

In this section, we have developed a more detailed understanding of the scheduling characteristics of the utility-based resource allocator. By varying the weight of an external credit-value metric relative to an intrinsic estimated response time and NTCT metric, we are able to observe the benefits of both metrics in the utility function. Whereas the value metric contributes to decreased efficiency, it is clear that high value tasks are preferred over low value tasks as the credit-value weight is increased. Further, the credit-value associated with a task correlates with the preference for the task, as indicated by the negative correlation between startup delay and priority. In summary, it is shown that $0.5 \leq w_v \leq 0.7$ result in a compromise between efficiency and value effectiveness.

In the NTCT analysis, we measured the effect of a mediating NTCT metric on an estimated response time metric. Our results showed that the introduction of a lightly-weighted NTCT term decreases the effect of late preemptions without a large decrease in efficiency.

Finally, we observed the effect of varying the shape of a sigmoidal normalizing utility function. By varying α, we observed the characteristics resulting from nearly-linear, non-saturating, and saturating utility functions. The benefit of a non-linear yet non-saturating utility function is clearly shown for this workload, with a peak

Table 5.1: PB Design with Fold Over for 7 Parameters having 16 Experiments

	Parameters						
	A	B	C	D	E	F	G
1	+	+	+	-	+	-	-
2	-	+	+	+	-	+	-
3	-	-	+	+	+	-	+
4	+	-	-	+	+	+	-
5	-	+	-	-	+	+	+
6	+	-	+	-	-	+	+
7	+	+	-	+	-	-	+
8	-	-	-	-	-	-	-
9	-	-	-	+	-	+	+
10	+	-	-	-	+	-	+
11	+	+	-	-	-	+	-
12	-	+	+	-	-	-	+
13	+	-	+	+	-	-	-
14	-	+	-	+	+	-	-
15	-	-	+	-	+	+	-
16	+	+	+	+	+	+	+

improvement in completion time being observed when $\alpha = 10$.

5.3 Plackett-Burman Design of the Utility/Knapsack Scheduler

In this section we use the Plackett-Burman design methodology [68] to improve our understanding of the utility/knapsack scheduler. We review the Plackett-Burman design methodology in section 5.3.1 and then apply the technique to the design of a utility/knapsack scheduler and a generalized utility function.

5.3.1 Plackett-Burman Designs

Consider a design having N parameters, each of which taking on one of L values. In order to perform an exhaustive design, it is possible to undertake a study consisting of L^N experiments. The optimal design is then characterized by the set of parameters

that correspond to the best target variable. For reasons of practicality, most designs choose between two values for each parameter ($L = 2$). However this still requires 2^N experiments, which quickly becomes infeasible for a moderate number of parameters.

Plackett and Burman introduced a method [68] to reduce the number of experiments in cases where multi-parameter interactions are not present. The PB design can adequately measure the effects of N parameters using X experiments, where X is the next multiple of 4 strictly greater than N. Using the PB design with fold over [70], single and two-parameters interactions are handled while requiring only $2X$ experiments.

An example matrix for a PB design with fold over is shown in table 5.1. This matrix corresponds to $X = 8$, and could be used for 4, 5, 6, or 7 parameter experiments. The columns of the matrix correspond to parameters, and the rows correspond to experiments. Each experiment (row) is configured using a series of high (+) and low (-) values for the parameters. The high and low values of a parameter are selected to be just outside the range of normal values. While the matrix for $X = 8$ is given here and used in this section, the first rows of the matrices for larger values of X can be acquired from [68]. The second and further rows are found by rotating the previous row to the right; the final (X-th) row is a series of low values. In the case of PB with fold over, a further X rows having opposite values is included, thereby making the total number of required experiments equal $2X$. The matrix can be verified by ensuring that the numbers of +'s and -'s in each column are each equal to X.

For each experiment, a target variable is measured and recorded. After running $2X$ experiments, it is possible to compute the *main effect* of each parameter by subtracting the target's values corresponding to the low parameter values from those

when it is high. For example, to compute the main effect of parameter B we use:

$$\begin{aligned} m_B = \quad & V_1 + V_2 + V_5 + V_7 + V_{11} + V_{12} + V_{14} + V_{16} \\ & -V_3 - V_4 - V_6 - V_8 - V_9 - V_{10} - V_{13} - V_{15} \end{aligned} \tag{5.6}$$

where V_i is the value of the target variable for experiment i. When comparing effects of different parameters, only the magnitude is used; the sign of the effect is not important. Before analyzing the results of a design, the effects of the parameters are ranked from 1 to $X-1$. This prevents any single parameter from dominating others, yet amplifies the difference between ones that are very similar.

It is important for designers to exercise diligence when using the PB design. There are a number of cases for which the methodology may not adequately show the effects of a parameter, thereby providing misleading results. Firstly, when the effect of the interactions of three or more parameters is large, the PB design should not be used. The PB design can find the effects of single parameters, and by doubling the number of experiments using a technique called fold over, two-factor interactions are allowed. Further, using only the high and low parameter values may hide effects in some situations. It is possible that an effect is large at a point between the high and low values, but the difference between effects at the end points is minimal. One way to solve this is to use 3 or more levels (low, medium, high) in the design.

5.3.2 PB Design of a Utility/Knapsack Metascheduler

In the preceding presentations of the utility/knapsack system, we evaluated a number of utility functions using a variety of benchmark task traces. This section focuses on the design of a single utility function u_{ij} which includes all of the QoS metrics which have been introduced. Specifically, the utility of the jth option of task i is given to

Table 5.2: High and Low Values for the Metascheduler Parameters

Parameter	- Value	+ Value
τ	3800	3400
w_v	.2	.8
w_{ERT}	.2	.8
w_{NTCT}	.2	.8
α_v	1	10
α_{ERT}	1	10
α_{NTCT}	1	10

be

$$\begin{aligned} u_{ij} &= w_v S\left(\alpha_v, \frac{v_{ij}}{v_{max}}\right) \\ &+ w_{ERT} S\left(\alpha_{ERT}, \frac{t - s_i + r_{ij}}{N_i}\right) \\ &+ w_{NTCT} S\left(\alpha_{NTCT}, \frac{N_i}{r_{ij}}\right) \end{aligned} \tag{5.7}$$

(5.8)

where the first term is a credit-value metric, the second term is an estimated response time (ERT) metric, and the third term is a nearness to completion time (NTCT) metric. Notation is used as follows: w_v, w_{ERT}, and w_{NTCT} are weighting factors, v_{ij} is the credit-value paid for a task-option, v_{max} is the largest credit-value, t is the current time, s_i is the submission time, r_{ij} is the estimated remaining time, N_i is the total computation length. Finally, $S(\alpha, x)$ is a monotonically increasing sigmoidal normalizing function given by

$$S(\alpha, x) = \frac{2}{\exp(-\alpha x) + 1} \tag{5.9}$$

An example sigmoidal function is shown in figure 4.1.

The parameters studied in this design are the reallocation period τ, the weights

of the three policy terms in the utility function, and the three α parameters, which are used to shape the sigmoidal utility functions. Having $N = 7$ parameters to study, the PB design with fold over for $X = 8$ (shown in table 5.1) can be used. (If N is not exactly $X - 1$, where X is a multiple of 4, then dummy parameters can be used.) The high and low values for each parameter is given in table 5.2. Each of the three term weights is taken to be either 0.2 or 0.8 in order to measure the effects when the terms are not heavily weighted and when they are more dominant. Also, each of the sigmoidal shaping values of α are 1 or 10, representing linear and saturating normalizing functions, respectively. For τ, the low value is 3800s and the high value is 3400s. Notice that the τ values are ordered contrary to their magnitude. This is because a smaller reallocation period improves the performance of the metascheduler.

5.3.3 Measuring the Main Effects

The experiments outlined above were carried out in a simulation of the utility/knapsack system. The simulation assumes the same grid as in section 4.5.2; the grid has four clusters of 16, 16, 32, and 128 processors each. The knapsack metascheduler is evaluated using task traces derived from the NPACI JOBLOG. The design is evaluated using 10 random offsets in the JOBLOG. For each of these random seeds, the PB design with fold over is performed and the parameters are ranked. When the studies of 10 seeds are completed, the average ranks are found and conclusions can be drawn about the relative importance of the metascheduler parameters.

The performance of the utility/knapsack strategy can be characterized using a number of measures. In this section, we select three measures. First, the resource utilization, or schedule efficiency is important to ensure that resources do not remain idle; this is indicated by the overall completion time of the simulated schedule. Second, the mean delay of tasks waiting in the queue should be as small as possible; this ensures that users are not waiting an unreasonably long time for their task to be executed. Finally, the correlation between this delay and the credit-value paid for

Table 5.3: Overall Completion Time Results of the Metascheduler Design (Seed 1)

	Parameters							Time
	τ	w_v	w_{ERT}	w_{NTCT}	α_v	α_{ERT}	α_{NTCT}	$(*10^6)$
1	+	+	+	-	+	-	-	3.326
2	-	+	+	+	-	+	-	3.417
3	-	-	+	+	+	-	+	3.469
4	+	-	-	+	+	+	-	3.391
5	-	+	-	-	+	+	+	3.419
6	+	-	+	-	-	+	+	3.323
7	+	+	-	+	-	-	+	3.413
8	-	-	-	-	-	-	-	3.451
9	-	-	-	+	-	+	+	3.483
10	+	-	-	-	+	-	+	3.340
11	+	+	-	-	-	+	-	3.309
12	-	+	+	-	-	-	+	3.386
13	+	-	+	+	-	-	-	3.374
14	-	+	-	+	+	-	-	3.429
15	-	-	+	-	+	+	-	3.406
16	+	+	+	+	+	+	+	3.362
Effect $(*10^5)$	-6.209	-1.747	-1.726	3.763	-0.153	-0.774	0.942	
Rank	1	3	4	2	7	6	5	

a task is important; users want to see that their large payment for an urgent task works to have it executed sooner.

Table 5.3 shows the full overall completion time results for the first random offset. For each of the 16 experiments, the overall completion time is shown in the rightmost column. The effect and rank of each parameter is shown in the bottom two rows. In this case, τ and w_{NTCT} are shown to be the most important parameter in ensuring timely completion of all of the NPACI tasks. Note that this does not imply the direction of these parameters' effects on the overall completion time; indeed, it is evident that the high value for τ leads to better performance, whereas the high value for w_{NTCT} leads to worse performance.

The overall completion time results for all 10 random offsets are summarized in table 5.4. For each random seed, the parameters are ranked, and for each parameter,

Table 5.4: Ranking the Design Parameter Effects on Overall Completion Time

	Random Seed										
Parameter	1	2	3	4	5	6	7	8	9	10	Mean
τ	1	1	1	1	1	1	1	1	1	1	1
w_v	3	2	3	2	7	2	2	5	7	3	3.6
w_{NTCT}	2	4	2	3	6	4	6	6	5	2	4
α_{NTCT}	5	7	4	7	2	7	4	2	2	4	4.4
α_v	7	3	5	6	4	3	3	4	6	7	4.8
w_{ERT}	4	5	6	4	3	5	5	7	4	6	4.9
α_{ERT}	6	6	7	5	5	6	7	3	3	5	5.3

Table 5.5: Ranking the Design Parameter Effects on Mean Queue Delay

	Random Seed										
Parameter	1	2	3	4	5	6	7	8	9	10	Mean
w_{ERT}	2	2	2	1	3	1	1	2	1	2	1.7
α_v	1	1	1	2	6	4	2	5	2	1	2.5
α_{ERT}	3	3	4	4	2	2	4	3	3	4	3.2
w_v	4	4	3	3	4	3	3	4	4	3	3.5
α_{NTCT}	5	5	5	5	5	5	5	6	6	5	5.2
τ	6	7	7	6	1	7	7	1	5	7	5.4
w_{NTCT}	7	6	6	7	7	6	6	7	7	6	6.5

the mean rank is shown. The table shows that in all cases the dominant parameter in determining schedule overall completion time is the reallocation period τ. This result becomes clear when the allocation strategy is considered. At allocation time, the metascheduler attempts to use all of the available resources by scheduling a task on each available processor. However, as tasks finish, the processors on which they were running become idle. It is not until the reallocation time that the idle processors are used again. Therefore, a smaller reallocation period (such as the "high" value) leads to a more efficient schedule.

The ranking results of the mean queue delay for the 10 seeds is shown in table 5.5. The two main factors influencing the delay are found to be w_{ERT}, the weight

Table 5.6: Ranking the Design Parameter Effects on Money-Delay Correlation

	Random Seed										
Parameter	1	2	3	4	5	6	7	8	9	10	Mean
w_{ERT}	2	1	2	1	2	2	1	1	1	1	1.4
w_v	1	2	1	2	1	1	2	2	3	2	1.7
α_{ERT}	3	3	4	3	3	3	4	3	2	3	3.1
α_v	4	4	6	4	6	4	3	7	4	5	4.7
w_{NTCT}	5	5	3	5	4	5	7	5	6	4	4.9
α_{NTCT}	7	6	5	7	7	6	6	6	5	6	6.1
τ	6	7	7	6	5	7	5	4	7	7	6.1

of the estimated response time term, and α_v, the sigmoidal normalizing function shaping value. The weight of the ERT is correctly influencing the delay more than the other parameters. The ERT metric grows larger as tasks wait to be scheduled for execution. This preference toward tasks that have been waiting long ensures that the mean delay stays low.

The influence of α_v is, however, surprising and requires discussion. The high value of α_v provides for earlier saturation of the credit-values. It is possible that the high value is encouraging a better delay because this saturation leads to similar utilities among many tasks. When options have similar utilities, the resulting allocations will become effectively random. The nearly equal preference of all tasks leads to a lower mean delay.

It is also notable that for two out of the ten seeds, τ was the most dominant parameter. It is likely that for these offsets the tasks were sporadic, and thus the reallocation rate influenced the delay significantly.

Finally, the ranking results for the correlation between credit-value and delay are shown in table 5.6. The correlation is expected to be larger in magnitude when the weight of the credit-value term is high. However, the ranking results show that both w_v and w_{ERT} play a large role in practise. When the metascheduler is performing

an allocation, many tradeoffs are made to find the best overall utility. Having an ERT term which is highly weighted leads to the highly valued tasks being delayed, decreasing the magnitude of the correlation between credit-value and delay. Further, a highly weighted credit-value term allows for highly valued tasks to be quickly scheduled, which increases the magnitude of the correlation. Clearly, both parameters have a large effect on the correlation metric and must be chosen carefully.

5.4 Conclusions

This chapter has presented two design and analysis techniques which were used to achieve a better understanding of the utility/knapsack scheduling strategy. First, sensitivity analyses were used to observe the effects of varying the weights of the QoS metrics in the utility functions. Next, a Plackett-Burman design with fold over was used to further understand the metrics' effects on target variables such as overall completion time and task startup delay.

The sensitivity analyses of the utility/knapsack strategy were performed by varying a number of parameters in the utility functions. Three analyses were performed. The first demonstrated that increasing the weight of a credit-value metric in a utility heuristic leads to a larger correlation between value and delay at the cost of longer overall completion time. The second showed that the the nearness-to-completion-time metric can be used to decrease the mean task execution time while only marginally increasing the overall completion time. The final analysis showed that a tuned sigmoidal utility function can decrease the overall completion time. In general, the results demonstrate that sensitivity analyses can be used to improve the overall performance of the utility/knapsack approach.

Finally, we presented a Plackett-Burman design of the utility/knapsack system. The analysis showed both how the PB design is effective in understanding the effects of the parameters, and leads toward further exploration of the significant parameters

to find the optimal design. It is notable that the PB methodology led to a new understanding of certain parameters on the performance of the system (e.g. the influence of α_v on the mean queue delay). Such relations were not apparent under the classical sensitivity analyses. We believe that the Plackett-Burman methodology has a significant role to play as we work towards optimizing the design of metaschedulers for the grid.

Chapter 6

Allocation of Multiple Resource Types

Computational grids commonly present a variety of resources to their users, including processors, storage, memory, and network bandwidth. One challenge in such systems is finding the optimal resource allocation for a set of tasks, each of which having unique requirements for the different resource types. In this chapter, we present a solution which employs a multiresource extension of the utility/knapsack technique introduced in chapter 4. We present utility heuristics as functions of multiple resource type metrics and solve the allocation using a 0-1 multichoice multidimensional knapsack problem. The technique is shown to perform well for a simulated grid of processors, storage, and network bandwidth.

This work presented in this chapter was published [71] in the Proceedings of Grid'06, the 7th IEEE/ACM International Conference on Grid Computing.

6.1 Introduction

Computational grids are distributed systems developed with the primary purpose of providing scalable processing capability to virtual organizations of users. However, in order to make use of this processing power, applications often have critical requirements of other resource types, such as random access memory, scratch disk storage, and network bandwidth. The utility/knapsack strategy presented in chapter 4 presents a solution to the allocation problem for grid processors, without consider-

ing the effects of these additional resource types.

In this chapter, we extend that single-dimensional solution to a general solution of the allocation of multiple grid resource types. We introduce QoS metrics related to multiple resources, and derive utility function heuristics based on them. Further, we show that a task's requirement of multiple resource types can be expressed as multidimensional resource constraints in a multichoice multidimensional knapsack problem (MMKP). Using a simulated grid with processors, disk storage, and network bandwidth, we show that the utility/knapsack approach can result in more efficient use of the grid and, like in the single dimensional case, the use of a credit-value metric allows users to intervene with urgent tasks.

The remainder of this chapter is organized as follows. An example multidimensional allocation problem is discussed in section 6.2. Section 6.3 formalizes the MMKP formulation and section 6.4 discusses multiresource policies and their utility function heuristics. In section 6.5 we present our simulation results and by comparing multiple allocation policies. Finally, we conclude in section 6.6.

6.2 Example Allocation Problem

Consider a computational grid of 2 clusters, each of which provides processor and storage resources to the grid users. Cluster A is composed of 64 processors and provides a total of 10 gigabytes of scratch storage to the grid users. Cluster B is composed of 16 processors and provides 15 gigabytes of scratch storage.

A grid metascheduler queues incoming tasks and makes allocation decisions. Consider a task queue that contains 2 malleable tasks. Each task lists a set of allocation options and is associated with a utility value. The first task specifies 3 options, and the second task specifies 2 options:

- Task 1, option 1: require 64 processors and 5 gigabytes storage on cluster A, utility 120.

- Task 1, option 2: require 32 processors and 5 gigabytes on cluster A, utility 110.
- Task 1, option 3: require 16 processors and 5 gigabytes on cluster B, utility 30.
- Task 2, option 1: require 64 processors and 10 gigabytes on cluster A, utility 150.
- Task 2, option 2: require 32 processors and 10 gigabytes on cluster A, utility 75.

Using these values, we can formulate the resource allocation problem as a multichoice multidimensional knapsack problem and by solving it we can find the allocation which maximizes the overall grid utility. We use the notation x_{ij} to indicate whether option j of task i is selected ($x_{ij} \in \{0, 1\}$). In this example, the selection of task 1 is limited to one of x_{11}, x_{12}, or x_{13}. Similarly, the task 2 options are indicated by x_{21} and x_{22}. In order to constrain the selection of each task to zero or one of its options, we introduce a multichoice constraint for each task:

$$x_{11} + x_{12} + x_{13} \leq 1 \tag{6.1}$$

$$x_{21} + x_{22} \leq 1. \tag{6.2}$$

There are also processor and storage resource constraints for each cluster. Cluster A is constrained to a total of 64 processors and 10 gigabytes of storage, thus we have the following constraints:

$$64x_{11} + 32x_{12} + 64x_{21} + 32x_{22} \leq 64 \tag{6.3}$$

$$5x_{11} + 5x_{12} + 10x_{21} + 10x_{22} \leq 10 \tag{6.4}$$

Similarly, cluster B is constrained to 16 processors and 15 gigabytes:

$$16x_{13} \leq 16 \tag{6.5}$$

$$5x_{13} \leq 15 \tag{6.6}$$

Finally, we form the objective function of the MMKP as the sum of the task option utilities:

$$f(x) = 120x_{11} + 110x_{12} + 25x_{13} + 150x_{21} + 75x_{22} \tag{6.7}$$

The MMKP formed therefore seeks to maximize equation 6.7 subject to the constraints defined in equations 6.1 through 6.6. Having few variables, the solution to this problem is trivial. By selecting x_{13} (task 1 option 3) and x_{21} (task 2 option 1), we reach the optimal value of 175. Notice that if the storage constraints had not been included, we could have selected x_{12} (task 1 option 2) and x_{22} (task 2 option 2), which would achieve a utility value of 185. However, this point is not valid in the constrained problem due to the storage requirements on cluster A not being met.

6.3 Allocating Multiple Resource Types using the MMKP

The knapsack formulation in chapter 4 is based around task options which include single dimensional resource constraints. By extending these constraints into multiple dimensions, we can express task options which require multiple resource types. We formulate the allocation of grid resources of multiple types as a 0-1 MMKP as follows:

$$\begin{aligned}
\text{Maximize} \quad & f(x) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} u_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{i=1}^{n} \sum_{j=1}^{m_i} a_{ijkl} x_{ij} \leq R_{kl} \\
& \sum_{j=1}^{m_i} x_{ij} \leq 1 \\
& x_{ij} \in \{0,1\} \text{ for } i \in \{1,...,n\} \\
& k \in \{1,...,r\} \\
& l \in \{1,...,\rho\}
\end{aligned} \tag{6.8}$$

Like in the single dimensional case , each task option has a corresponding utility value u_{ij}, which is computed using a utility function $U(\cdot)$ of intrinsic QoS metrics. However, in this case, each option has a corresponding set of resource demands a_{ijkl} for each resource centre k (of r in total) and resource type l (of ρ in total). Further, each resource centre k is constrained to the resource availability R_{kl} for each of the resource types l. Finally, x_{ij} indicates whether or not task i option j is selected.

6.4 Multi-Resource Policies and Utility Function Heuristics

The importance of the utility/knapsack approach is that it allows grid administrators to define allocation policies by quantifying an allocation option's perceived utility, computed as a function of intrinsic and external QoS metrics. Chapter 4 introduced a number of allocation policies and their associated utility functions. Specifically, the QoS metrics used in these functions include the credit-value metric CV_{ij}, which allows users to spend virtual money on task options according to their task priorities, the estimated-response-time metric ERT_{ij}, which estimates the remaining normalized running time of a task on a particular cluster, and the nearness-to-completion-time metric $NTCT_{ij}$, which increases as tasks near their finishing time. These metrics are defined for a particular option j of task i as

$$CV_{ij} = v_{ij} \tag{6.9}$$

$$ERT_{ij} = \frac{t - s_i + r_{ij}}{N_i} \tag{6.10}$$

$$NTCT_{ij} = \frac{N_i}{r_{ij}} \tag{6.11}$$

where v_{ij} is the assigned credit-value, t is the current time, s_i is the submitted time, r_{ij} is the task's remaining time under option j, and N_i is the total task length. Using these metrics, it is possible to define a utility function which places equal preference

on CV_{ij}, ERT_{ij}, and $NTCT_{ij}$:

$$u_{ij} = S(CV_{ij}) + S(ERT_{ij}) + S(NTCT_{ij}) \tag{6.12}$$

where $S(\cdot)$ is a non-decreasing positive normalizing function. Note that chapter 5 demonstrated that placing equal weights on the CV_{ij}, ERT_{ij}, and $NTCT_{ij}$ metrics leads to a well-rounded utility function.

6.4.1 Storage and Network Resource Metrics

We now introduce new metrics which leverage the multiple resource types of a computational grid. Consider that a grid application typically transfers some input data of size d_{ij} from a central storage server before the process can begin. The network bandwidth between the cluster k and the storage server is B_k. The cluster employed by option j of task i is determined by a mapping function

$$k = \text{clustermap}(i, j) \tag{6.13}$$

therefore we denote

$$B_{ij} = B_{\text{clustermap}(i,j)}. \tag{6.14}$$

It is generally preferable to assign a task to the cluster at which it can retrieve its input data the fastest. To quantify this policy, we can derive an input data transfer metric IDT_{ij} as

$$IDT_{ij} = \frac{B_{ij}}{d_{ij}} \tag{6.15}$$

For clusters having a larger bandwidth B_k, the IDT_{ij} is large, indicating a preference for clusters with a fast network connection to the storage server. Also, smaller amounts of input data d_{ij} will increase the IDT_{ij}, thereby preferring the tasks with small data requirements.

In order to incorporate the IDT_{ij} metric, while maintaining the desirable schedule characteristics created by the CV_{ij}, ERT_{ij}, and $NTCT_{ij}$ metrics, we use the following utility function:

$$u_{ij} = S(CV_{ij}) + S(ERT_{ij}) + S(NTCT_{ij}) + S(IDT_{ij}) \tag{6.16}$$

6.5 Experimentation

The performance of the multiresource utility/knapsack strategy has been evaluated using a grid simulation developed using the SimGrid Toolkit [64]. The simulated grid features four clusters with 16, 16, 32, and 128 homogeneous processors, and 100, 300, 400, and 1024 gigabytes of scratch storage, respectively. The bandwidth between each cluster and a central input data storage facility is a random value between 10 and 100 megabits per second; the bandwidth is fixed for each simulation experiment.

The task workload is generated by extracting sequences of 1000 tasks at random offsets from the NPACI JOBLOG; 5 such workloads are simulated and mean results are presented. Task input data sizes are uniformly distributed between $100d$ and $1000d$ megabytes, where d is a data size scaling factor. In order to observe a full range of data sizes, we scale d from 1 to 1000; at the low end, the data sizes are well below the resource constraints, and at the high end, the data sizes severely limit the number of tasks simultaneously allocated to each cluster.

In the case of task migrations, we assume that all of a task's input data on a cluster is deleted when the task is migrated away from that cluster. Thus, migration from one cluster to another incurs the full input data transfer time. Similarly, in the case that a task is reallocated to a cluster on which it is currently processing, there is no input data transfer time required.

The allocation strategy presented in this dissertation relies on periodic reoptimization of the knapsack problem every τ seconds. The single dimensional problem

presented in chapter 4 used large values for τ in order to reduce the effects of reallocation overhead. However, in this multidimensional solution, the overhead of moving data is factored into the utility function heuristics. We therefore have configured the simulations with frequent reallocations, every $\tau = 255.2$s; this rate is equal to the mean task interarrival rate λ, so it is ensured that tasks will not wait in the queue while resources sit idle.

6.5.1 Simulated Allocation Strategies

The flexibility of the utility/knapsack approach can be demonstrated by evaluating a variety of allocation policies and their corresponding utility functions. Additionally, it is important to measure the performance of the proposed scheduler against existing strategies. Accordingly, we have incorporated the scheduling results of two reference strategies:

Basic FCFS Tasks are executed in the order they arrive in the queue. Processor and disk constraints are obeyed.

Data FCFS Tasks are executed in the order they arrive in the queue; however, each task's options are sorted in ascending order of estimated data transfer time plus the remaining processing time. This local strategy prefers the fastest cluster in terms of both processor and storage resources. It does not provide any global optimization. Again, processor and disk constraints are obeyed.

Using the utility/knapsack allocation formulation, we evaluate a variety of utility functions and measure their effects on the resulting schedules:

UM Data $u_{ij} = S(ERT_{ij}) + S(NTCT_{ij}) + S(IDT_{ij})$.

UM NonData $u_{ij} = S(ERT_{ij}) + S(NTCT_{ij})$.

UM Data+Money $u_{ij} = S(CV_{ij}) + S(ERT_{ij}) + S(NTCT_{ij}) + S(IDT_{ij})$, where CV_{ij} is a uniformly distributed random value between 0 and 1.

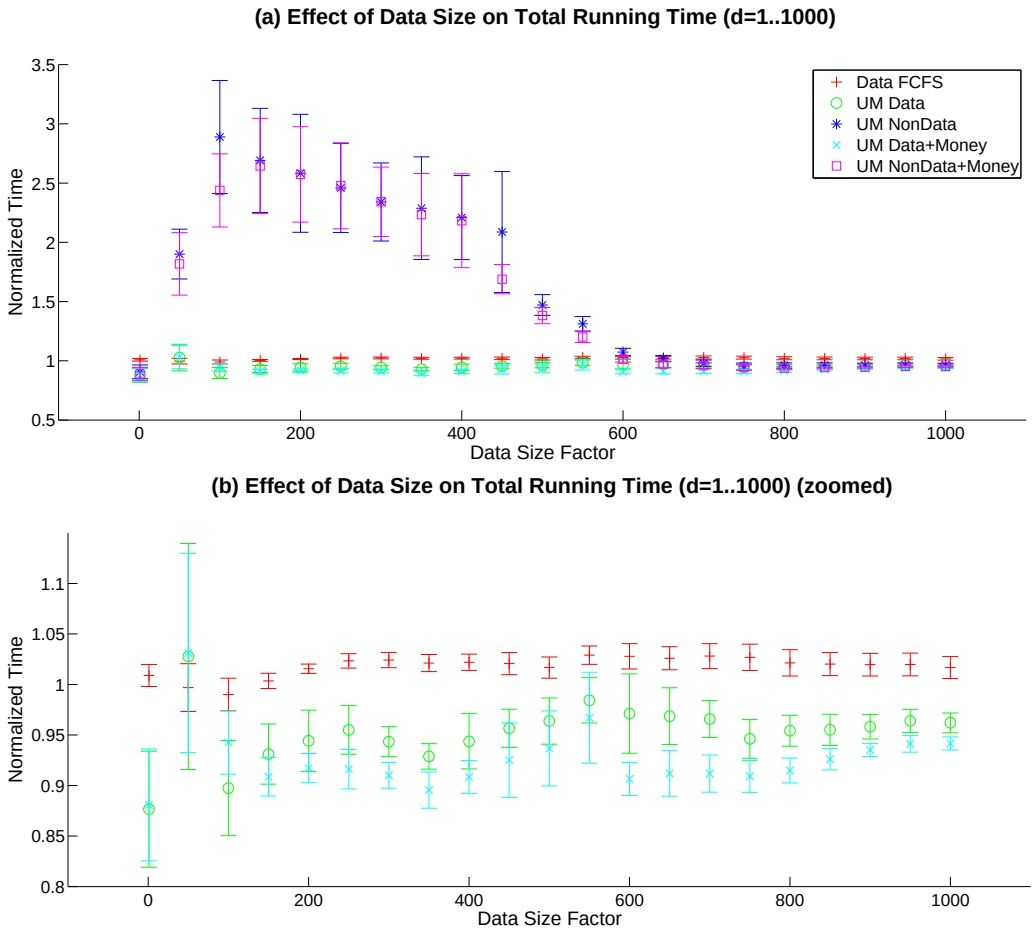

Figure 6.1: Total running time normalized to that of Basic FCFS for all strategies and (a) all policies (data-aware and non-data-aware); and (b) only the data-aware policies, for clarity.

UM Non-Data+Money $u_{ij} = S(CV_{ij}) + S(ERT_{ij}) + S(NTCT_{ij})$, where CV is a uniformly distributed random value between 0 and 1.

Note again that we have used equal weighting factors on all of the above QoS metrics. This was shown in chapter 5 to result in well-rounded utility functions.

6.5.2 Results and Discussion

The simulation results are presented in figures 6.1, 6.2, and 6.3. Tables A.5, A.6, and A.7 present the complete data results for reference.

Before observing the figures, it is useful to note one counterintuitive result of the simulations. It may be expected that the total running time should increase linearly

as the data size factor d increases. However, this is not the case; as d increases, the running time increases faster than linearly. The running time does increase linearly with d within narrow regions; the nodes between these linear sections occur at points at which the storage constraints limit which clusters can be used. For example, the cluster having 16 processors and 100 gigabytes of storage can no longer accept a full allocation of tasks at values for $d > 64$ (above this point it is impossible to fit 16 of the smallest tasks due to the disk constraint). At very large values of d, the storage constraints severely limit which clusters can be used.

Figure 6.1 (table A.5) shows the total running times of all tasks normalized to that of Basic FCFS. In figure 6.1 (a), we observe that UM NonData and UM NonData+Money result in dramatic increase in the total running time for the middle values of d. Because these policies do not consider the data movement overhead when a task is migrated to another cluster, optimizing the NonData utility functions leads to frequent migrations, and since each migration results in a data migration, the total running time is increased significantly. As d becomes larger, the number of clusters that can meet the storage constraints decreases, leading to a decreased opportunity for migrations; this limitation on the available clusters leads to more competitive times for the NonData policies. The transfer times observed later in figure 6.2 verify this interpretation of the results.

In figure 6.1 (b), we isolate the data-aware policies. It is notable that the Data FCFS policy does not perform better overall than Basic FCFS; considering data alone will not lead to an improvement over Basic FCFS. On the other hand, UM Data and UM Data+Money, do improve on Basic FCFS for almost all data loads, achieving a 2-11% increase in performance. At the data loads corresponding to $30 \leq d \leq 70$, these policies have an outlying performance roughly equal to Basic FCFS but with a large variance in the observed results. The long tail of the NPACI workload is creating challenges for the basic utility heuristic employed in this chapter; design

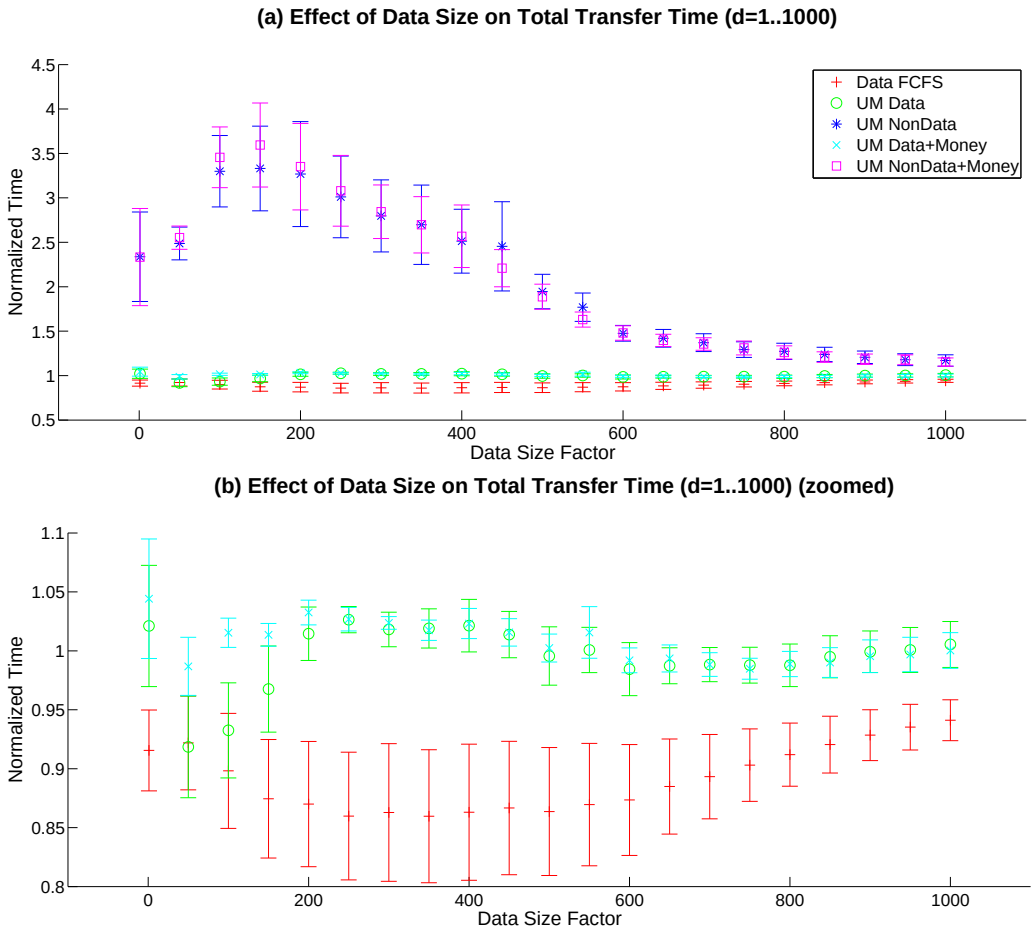

Figure 6.2: Total transfer time normalized to that of Basic FCFS for all strategies and (a) all policies, (b) just the data-aware policies.

procedures such as those in chapter 5 should be employed to optimize the utility function.

We have isolated the time spent transferring input data in figure 6.2 (table A.6). Data FCFS consistently spends the least time transferring data; this is because it always selects the cluster with the fastest bandwidth, and because it never migrates tasks, which would result in subsequent transfers. In addition, UM NonData and UM NonData+Money perform consistently poor; without considering the data transfer overhead of migration, they result in 2-3 times as much time spent transferring data. However, the incorporation of the IDT metric in UM Data and UM Data+Money eliminates this problem. For very small and very large values of d, the time spent

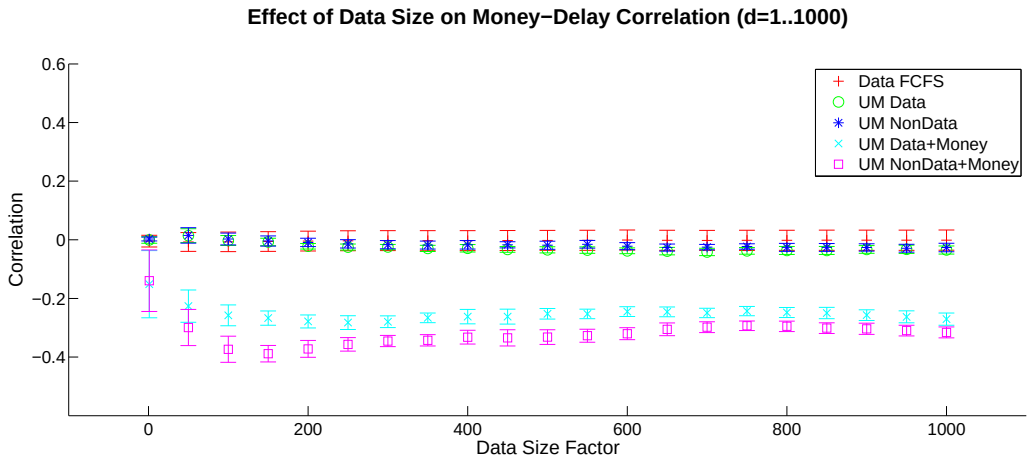

Figure 6.3: Correlation of offered credit-value with startup delay.

transferring data is roughly equivalent to the Basic FCFS policy; because the combined ERT+NTCT, and ERT+NTCT+CV metrics outweigh the IDT metric, the optimal data location is not always selected in these cases. For $10 \leq d \leq 100$, however, UM Data and UM Data+Money achieve the optimal data locations. It is clear that inclusion of the IDT term is essential in this environment; indeed, when it is included the transfer time is reduced as intended.

Finally, we demonstrate the effect of the credit-value, or money, metric in figure 6.3 (table A.7). An increase in the credit-value offered for an option indicates a user's preference, and this should be reflected in the resulting schedule by seeing a decrease in the task delay (the start time minus the submitted time). This behaviour is observed as a negative correlation between credit-value and startup delay. Data FCFS, UM Data, and UM NonData result in no correlation for all values of d; these policies do not consider credit-value in their allocation decisions. However, when the CV metric is introduced as in UM Data+Money and UM NonData+Money, we observe a negative correlation. The consistent negative correlation in these strategies indicates that they are indeed able to preferentially select high valued options. Finally, when comparing UM Data+Money and UM NonData+Money, it is clear that the incorporation of the IDT metric leads to a decrease in correlation's magnitude; there is a

clear tradeoff between data transfer time and value/delay correlation.

6.6 Conclusions

In this chapter we have presented an extension of the utility/knapsack grid scheduling system to the allocation of multiple resource types. We have demonstrated the constraints of multiple resource types can be realized using the MMKP, and that QoS policies can be derived and implemented from their properties. In particular, we demonstrated a grid of processors, storage, and network resources and introduced utility function heuristics which result in schedules that preferentially allocate resources according to the credit-value, estimated-response-time, nearness-to-completion-time, and input data transfer QoS metrics. By simulating the scheduler, we demonstrated the the utility/knapsack strategies out-perform reference strategies in total running time and transfer time. In addition, the incorporation of credit-value in the utility results in a negative correlation between money offered and task startup delay.

Chapter 7

Selection between Fault Tolerance Options using the Utility/Knapsack Approach

This chapter presents an application of the utility model and knapsack solution to the challenge of selecting between fault tolerance techniques. On a typical grid, there are many ways to handle failed tasks, including retrying, replication, and checkpointing. However, given the variability in the urgency of the tasks and the reliability of the resources, it is not always apparent which of these techniques should be used for each task. By employing the utility model, we characterize each task's need for fault tolerance and the knapsack solution finds an optimal mapping of tasks to techniques.

This work was published in the Proceedings of the IEEE Conference on e-Science and Grid Computing 2007 [72].

7.1 Introduction

One problem common to most grids is the prevalence of faults, which in many cases are caused by the relative immaturity of grid middleware combined with the asymmetric configuration and reliability of the grid resources [73]. As such, a task and resource management system which can gracefully recover from or work around faults

is paramount to a successful grid user experience.

Grid designers typically implement one or more of three basic techniques to provide task-level fault tolerance [74]. The first technique, *retry*, automatically resubmits tasks that have failed. The *replication* technique submits multiple replicas of a task to the grid; if at least one of the replicas finishes successfully, the task is complete. The third technique makes use of task *checkpointing* to roll back failed tasks to a known-good state when a fault occurs. Each of these techniques varies in implementation complexity, resource cost, resilience, and expected response time. For example, replication requires more resources than retrying, yet in the case of faults the response time of the task does not increase. In the case of grids which support more than one fault tolerance technique, it is clear that the importance or urgency of a task should influence the technique employed. However, deciding which tasks employ which technique is complex, involving many factors that are usually unknown to users.

In this chapter, we present a technique for automatically matching tasks to fault-tolerant techniques using task, user, and grid metrics to compute the relative merits of each of the options. We compute the *utility* of each allocation option and then formulate as a knapsack problem to find the globally optimal allocation set. We develop three heuristics which take into account the value offered by a user, the estimated resource cost, and the estimated response time of an option. It is shown that the utility model effectively selects from a number of fault tolerance techniques to decrease running time, increase profit, while allowing users to prioritize their tasks.

The remainder of this chapter is organized as follows. Section 7.2 states our assumptions and defines our fault and failure model. In section 7.3 we formulate the problem using allocation option utility and show how to optimize the global utility. Next, we introduce utility heuristics for selecting fault tolerance techniques in section 7.4. Section 7.5 shows our experimentation results using a first-come-first-

serve scheduler as a reference strategy. Finally, in section 7.6 we reflect on the results and conclude.

7.1.1 Related Work

Due to the prevalence of faults on the grid [73], providing task-level fault tolerance has received a significant amount of research efforts. In their taxonomy on grid workflow management systems, Yu and Buyya classified task-level fault tolerance techniques into retry, alternate resource, checkpoint/restart and replication [74]. They show that due to the simplicity of its implementation, most of these systems employ the retry or alternate resource techniques, yet few implement checkpointing or replication. However, more theoretical work is being performed to study replication [75, 76] and checkpointing [76]. Other work has focussed on making the execution sites highly available through redundancy, effectively implementing failover via replication [77]. In contrast with these studies, our work recognizes that all fault tolerance techniques have varied trade-offs with regard to performance and cost, and seeks to find optimal mappings of tasks to techniques in complex grid environments.

7.2 Preliminaries

It is assumed that a grid is a distributed heterogeneous system composed of many resource centers (clusters). Users submit tasks to a resource allocation service, which in turn places the tasks on one or more of the clusters. The allocation service can be centralized or distributed, depending on its implementation.

When a task is placed onto a cluster, there is a probability that a fault will occur during its execution. Faults originate from the physical hardware, the network, the operating system, or from the task itself. Whereas most faults typically follow a Weibull distribution [78], in this work we simplify by encapsulating all sources of fault into a single measure which represents the probability that a given task will incur a fault on a given cluster. This measure can be practically forecasted using

tools such as the Network Weather Service [79]. Faults which are not handled by using one of the fault tolerance techniques result in the failure of a task.

There are two fault tolerance techniques incorporated into our selection heuristics: retry and replication. With the retry technique, we resubmit a faulty task to the same resource on which it was executing. With replication, we simultaneous place replicas of a task on distinct processors; it is improbable for all replicas of a task to be faulty. Finally, we allow tasks to request no fault tolerance. We have not included checkpointing in this study for two reasons. First, the primary focus of this work is to introduce and evaluate the utility-based selection methodology, not to evaluate viability of all of the possible fault tolerance techniques on the grid. Second, checkpointing has many parameters, such as checkpointing interval and overhead, which would need to be estimated for simulation; the wide range of values and large uncertainties on these parameters would reduce our statistical confidence in the results.

7.3 Intelligent Selection of Fault Tolerance Techniques

Consider a grid composed of r clusters and capable of providing task-level fault tolerance using one of m techniques. A resource allocation service has a task queue with n unplaced tasks. Because tasks can be allocated to any of the clusters using any of the fault tolerance techniques, there are mr placement options for each task. This problem is equivalent to the allocation problem formulated in chapter 4, assuming only grid processors are considered. In this case, the number of options for task i is $m_i = mr$, so the problem can be formulated as

$$\begin{aligned} \text{Maximize} \quad & f(x) = \sum_{i=1}^{n} \sum_{j=1}^{mr} u_{ij} x_{ij} \\ \text{subject to} \quad & \sum_{i=1}^{n} \sum_{j=1}^{mr} a_{ijk} x_{ij} \leq R_k \\ & \sum_{j=1}^{mr} x_{ij} \leq 1 \\ & x_{ij} \in \{0, 1\} \\ & k \in \{1, ..., r\} \end{aligned} \tag{7.1}$$

As in the previous chapters, the strength of this strategy lies in the selection of the utility functions used to compute u_{ij}. In the next section we introduce three heuristics that can be used to evaluate the relative merits of the various fault tolerance techniques.

7.4 Utility Functions

In this section, we address the problem of selecting an ideal combination of fault tolerance techniques by deriving three separate allocation heuristics and their associated utility functions. For each heuristic, we show that the utility of each option depends not only on the metrics of the task, but on the type of fault tolerance employed.

If P_N is the probability that a given task will be subjected to N faults, then

$$\sum_{N=0}^{\infty} P_N = 1. \tag{7.2}$$

Then, if U_N is the utility component in the case of N faults, we can define utility as the sum

$$u_{ij} = \sum_{N=0}^{\infty} P_N U_N. \tag{7.3}$$

For simplicity, we assume that tasks will be subjected to at most one fault; that is, the probability of two or more faults is zero. This assumption implies that tasks which use the retry technique will be retried at most once, and that at least one of a replicated task will complete successfully. Since the probability of a fault is usually

small ($\lesssim 10\%$), this first-order approximation will not contribute a large error to the final results. We therefore define the utility in each heuristic below to be the sum of the utility components in the cases of zero and one faults:

$$u_{ij} = P_0 U_0 + P_1 U_1 \tag{7.4}$$

where

$$P_0 + P_1 = 1. \tag{7.5}$$

The equations in the following sections use f_{ij} to denote the probability that option j of task i will fail, therefore

$$P_0 = 1 - f_{ij} \tag{7.6}$$

$$P_1 = f_{ij}. \tag{7.7}$$

Practical methods for estimating the likelihood of a fault is beyond the scope of this work. However, tools such as the Network Weather Service [79] have been shown to provide useful performance and fault forecasts [80] in related research.

7.4.1 Value Heuristic

The first heuristic equates the utility of a given task option to the expected *Value* should that option be selected. The offers made for each option are rational in most cases: users tend to pay more for longer, more urgent tasks, as well as for a completion guarantee provided by a fault-tolerant allocation. As such, a user's preferred allocation option within a task, their most urgent tasks in the queue, and their relative urgency in comparison to other users' tasks can all be inferred from the values offered. Further, the Value heuristic has the goal of optimizing the grid revenue for a single allocation cycle. Resources are allocated to whoever will pay the most; the performance and efficiency of the resulting tasks are not considered.

Let v_{ij} be the value offered by a user for the successful completion of task i under option j. Since users are not charged for failed tasks, the expected value of a task option varies depending on whether the option provides a completion guarantee. When no fault tolerance is selected, the utility components are $U_0 = v_{ij}$ and $U_1 = 0$ for zero and one faults, respectively. Therefore, the utility is

$$\begin{aligned} u_{ij} &= (1 - f_{ij})v_{ij} + f_{ij}(0) \\ &= (1 - f_{ij})v_{ij}. \end{aligned} \tag{7.8}$$

The retry and replication techniques guarantee completion, so the utility components are $U_0 = U_1 = v_{ij}$, and the utility is

$$\begin{aligned} u_{ij} &= (1 - f_{ij})v_{ij} + f_{ij}v_{ij} \\ &= v_{ij}. \end{aligned} \tag{7.9}$$

It is notable that the implementation of equations 7.9 and 7.8 results in a preference for the fault-tolerant techniques. This ensures that if a user has no preference and offers an equal value for all options, the fault tolerant options will be selected to guarantee revenue for executing the task.

7.4.2 Value/Cost Heuristic

The second heuristic maintains each option's estimated value, yet also considers the estimated resource cost to be incurred if a given option is selected. The estimated resource cost is defined as the estimated running time multiplied by the number of processors used. Running time can be estimated using a number of techniques, including simulation of the execution environment or using online learning from historical data [74]. The *Value/Cost* heuristic recognizes that the various fault tolerance techniques will consume an unequal amount of resources. Specifically, the retry tech-

nique will incur a further cost in the case of a fault, and replication always consumes twice the resources as the original task.

The resource cost is estimated by predicting the running time of a task, with consideration made to the resource's performance and faultiness. Given that L_i is a task's estimated running time on a reference processor, then the estimated running time on a processor with relative performance p_k is L_i/p_k. The cluster used by option j of task i is determined by a mapping function $k = \text{clustermap}(i, j)$. We therefore denote the estimated running time L_{ij} of option j of task i to be

$$L_{ij} = L_i / p_{\text{clustermap}(i,j)}. \tag{7.10}$$

As in the Value heuristic, we define f_{ij} to be the probability of a task to have a fault on a given cluster.

In the case of zero faults the estimated resource cost for both no fault tolerance and the retry technique is L_{ij}, whereas employing a single replica incurs a total cost of $2L_{ij}$. We now consider the case of a single fault. Assuming that faults are equally likely to occur at any point during an execution, the mean time of a fault is at the tasks midway point, $L_{ij}/2$. Thus, if a single fault were to occur, then the resource cost for no fault tolerance is $L_{ij}/2$, for the retry technique is $L_{ij}/2 + L_{ij} = 3/2L_{ij}$, and for replication is $2L_{ij} - L_{ij}/2 = 3/2L_{ij}$.

Using these estimated resource costs, combined with the option values, we can find the utility functions as follows. When no fault tolerance is provided, the utility is

$$\begin{aligned} u_{ij} &= (1 - f_{ij})\frac{v_{ij}}{L_{ij}} + f_{ij}(0) \\ &= (1 - f_{ij})\frac{v_{ij}}{L_{ij}} \end{aligned} \tag{7.11}$$

For the retry technique, utility is

$$\begin{aligned} u_{ij} &= (1 - f_{ij})\frac{v_{ij}}{L_{ij}} + f_{ij}\left(\frac{v_{ij}}{3/2L_{ij}}\right) \\ &= \frac{(3 - f_{ij})v_{ij}}{3L_{ij}} \end{aligned} \tag{7.12}$$

Finally, replication results in a utility of

$$\begin{aligned} u_{ij} &= (1 - f_{ij})\frac{v_{ij}}{2L_{ij}} + f_{ij}\left(\frac{v_{ij}}{3/2L_{ij}}\right) \\ &= \frac{(3 + f_{ij})v_{ij}}{6L_{ij}}. \end{aligned} \tag{7.13}$$

7.4.3 Value/ERT Heuristic

The third and final heuristic, *Value/ERT*, takes into account the estimated response time (ERT), that is, the time that a user is expected to wait for their task to complete successfully. It is clear that shorter response times, regardless of how the task is allocated, will increase user satisfaction.

In the case of zero faults, all fault tolerance techniques result in the same ERT, L_{ij}. If a single fault were to occur, the retry technique would respond in $L_{ij} + L_{ij}/2 = 3/2L_{ij}$, whereas replication will always respond within L_{ij}.

Again incorporating the offered value v_{ij}, we obtain the utility functions. For no fault tolerance, the utility is

$$u_{ij} = (1 - f_{ij})\frac{v_{ij}}{L_{ij}} \tag{7.14}$$

Table 7.1: Utility functions for the Value, Value/Cost, and Value/ERT heuristics

Heuristic	No FT	Retry	Replication
Value	$(1-f_{ij})v_{ij}$	v_{ij}	v_{ij}
Value/Cost	$(1-f_{ij})\frac{v_{ij}}{L_{ij}}$	$\frac{(3-f_{ij})v_{ij}}{3L_{ij}}$	$\frac{(3+f_{ij})v_{ij}}{6L_{ij}}$
Value/ERT	$(1-f_{ij})\frac{v_{ij}}{L_{ij}}$	$\frac{(3-f_{ij})v_{ij}}{3L_{ij}}$	$\frac{v_{ij}}{L_{ij}}$

For the retry technique, utility is

$$\begin{aligned} u_{ij} &= (1-f_{ij})\frac{v_{ij}}{L_{ij}} + f_{ij}\left(\frac{v_{ij}}{3/2L_{ij}}\right) \\ &= \frac{(3-f_{ij})v_{ij}}{3L_{ij}} \end{aligned} \tag{7.15}$$

Finally, replication results in a utility of

$$\begin{aligned} u_{ij} &= (1-f_{ij})\frac{v_{ij}}{L_{ij}} + f_{ij}\left(\frac{v_{ij}}{L_{ij}}\right) \\ &= \frac{v_{ij}}{L_{ij}} \end{aligned} \tag{7.16}$$

The utility functions for the Value, Value/Cost, and Value/ERT heuristics are summarized in table 7.1 for reference.

7.5 Experimentation

The placement algorithm and heuristics described above have been simulated using the SimGrid toolkit [64]. The simulation setup and parameters are shown in table 7.2. The experimentation is designed to evaluate the performance of the heuristics for varied levels of resource contention. In each simulation, we initialize a random task queue of length n and periodically perform an allocation as described in section 7.3 until all of the tasks have completed. Reallocations are performed every $\tau = 100$s in order to select the set of task options that maximize the global utility for the

Table 7.2: Simulator configuration and parameters

Parameter	Definition	Value
τ	Reallocation Period	100s
r	# Clusters	4
R_k	# Cluster Processors	[16 16 32 128]
p_k	Cluster Performance	Uniform RV over [0.5,1.5]
L_i	Task Lengths	Uniform RV over [0,86400]s
v_{ij}	Task Values	Uniform RV over [0,Task Length]
f_{ij}	Task-Cluster Fault Prob.	Normal RV, mean=0.1
m	# FT Techniques	3
u_{ij}	Utility Heuristics	Value, Value/Cost, Value/ERT
n	Queue Length	[10 20 40 80 160 320 640]
	# Random Seeds	1000

available resources. All tasks are assumed to have arrived at time 0. For each utility heuristic and queue length combination, we simulate 1000 random sets of tasks. The simulated grid has four clusters, each of which can execute any of the submitted tasks. Each task-cluster combination is associated with a random probability f_{ij} of a fault occurring, where f_{ij} is positive and normally distributed around 0.1. Faults occur at a uniformly random time during a task's execution. Each task can be allocated without fault tolerance or using one of the two described fault tolerance techniques: retry and replication. Thus, each task has 12 options in total (4 clusters * 3 fault tolerance techniques).

For simplicity, faults which occur when a task is allocated with neither retry nor replication result in a failure. Revenue is not earned from tasks that fail. When a fault occurs for a retry-enabled task, it is immediately re-executed on the same cluster. Also, no task can be subject to more than one fault; thus, both retried and replicated tasks do not fail. This simplification results in 1% fewer failures than the correct case, when tasks can be subject to multiple faults. As previously mentioned, the focus of this work is on the demonstration of the utility-based selection methodology. We intend to evaluate using a more accurate fault model in future work.

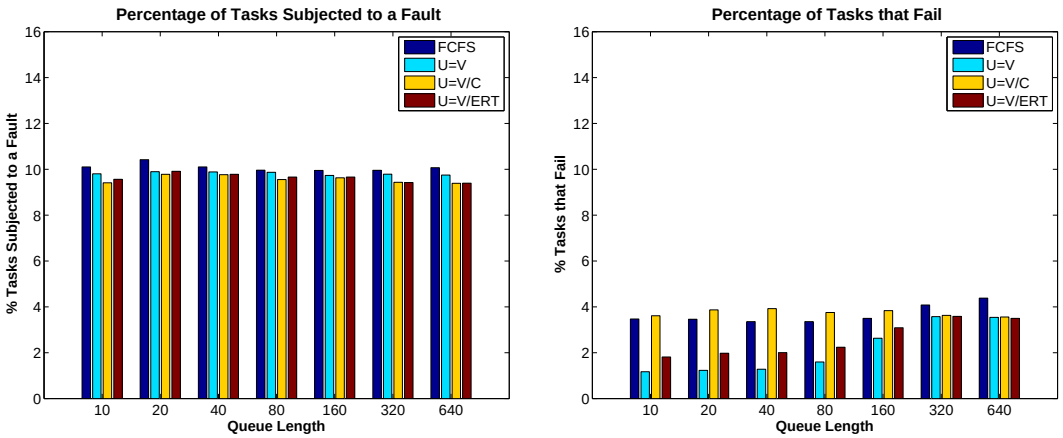

Figure 7.1: Mean percentage of tasks in the queue that are subjected to faults (left) and result in a failure (right)

The placement algorithm is evaluated using three heuristics: Value (U=V), Value/ Cost (U=V/C), and Value/ERT (U=V/ERT). For reference purposes, we have also simulated a first-come-first-serve (FCFS) strategy which allocates tasks in their order of arrival to the queue. In this strategy, each task is allocated using one of its options corresponding to a free resource. None of the value, length, fault probability, or performance metrics are used by the FCFS strategy. As such, the FCFS strategy will not have a preference for any of the fault tolerance techniques; it will allocate roughly one third to each.

7.5.1 Experimentation Results

Simulation results are presented in figures 7.1 through 7.4 and in table 7.3.

Figure 7.1 (left) presents the mean percentage of tasks that were subjected to a fault during the simulations. The simulator was configured to produce faults in roughly 10% of the tasks – this behaviour is shown. However, the heuristics are subjected to slightly fewer faults than the reference strategy. This is a result of the negative weight placed on the fault probability f_{ij} in the utility functions. The scheduler is preferring options which have a low fault probability.

In figure 7.1 (right), we can observe the mean percentage of tasks that resulted

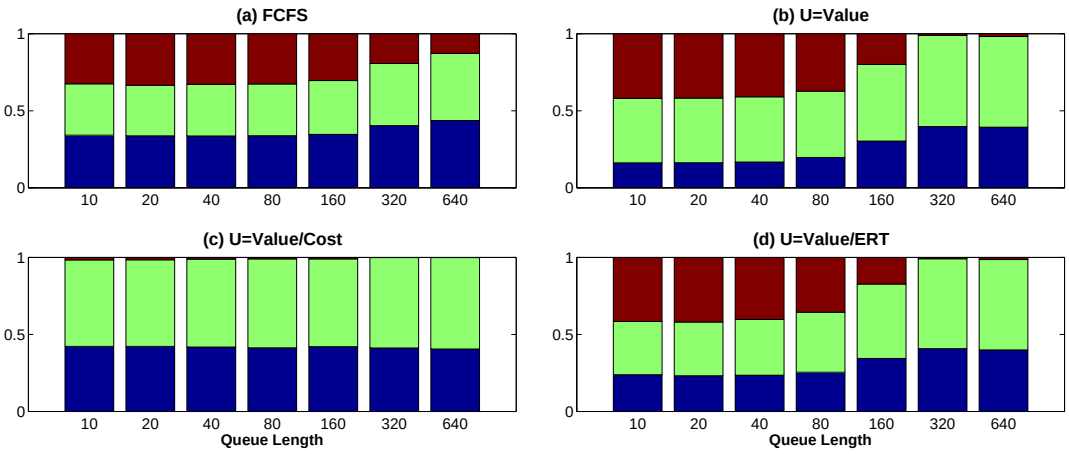

Figure 7.2: Relative proportion of fault tolerance techniques employed by each heuristic. Blue values correspond to no fault tolerance, green are retry, and red are replication.

in a failure. Failures result from faults that are not handled by one of the fault tolerance techniques. The reference FCFS strategy performs sub-optimally for all queue lengths. Because FCFS matches roughly a third of the queue to each fault tolerance technique, one third is placed without fault tolerance and therefore this proportion of the faults result in failures. Among the heuristics, Value results in the fewest number of failures for all queue lengths except 640. Value/ERT results in slightly more failures than Value, and at 640 it performs better than Value. Finally, Value/Cost results in more failures than FCFS for queue lengths less than 160 and below, however at 320 and 640 it performs better than FCFS.

To better understanding how the heuristics achieve these failure rates, we observe figure 7.2. This figure presents the relative proportion of fault tolerance techniques employed by the each heuristic. The number of failed tasks relates directly to the proportion of tasks selected to not use fault tolerance. For FCFS, we observe the expected one third ratio assigned to each technique when there are less than 160 tasks; however, for 160 tasks and more, we observe that resource contention leads to decreased replication availability. For Value and Value/ERT, the decreased expected value from options without fault tolerance ensures that not many are selected. In

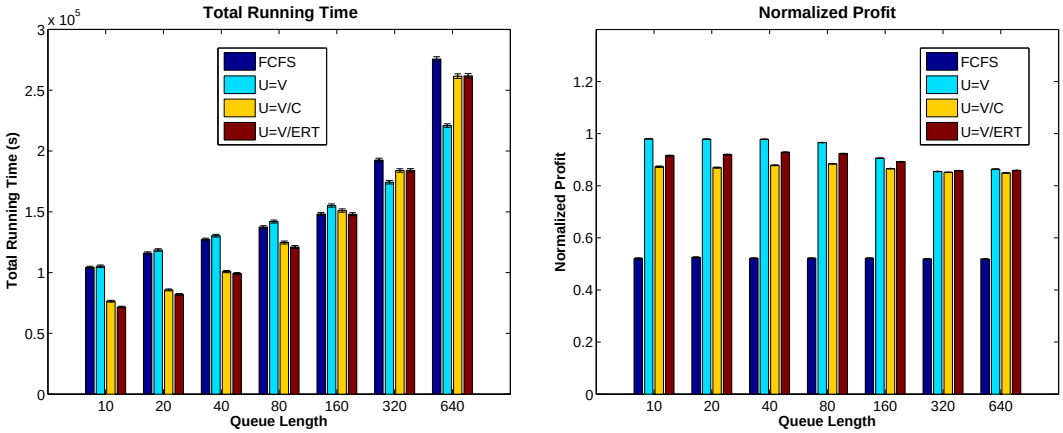

Figure 7.3: Mean total running time (left) and earned profit normalized to the available profit (right)

cases where no fault tolerance is selected, it is because it was not desired, i.e. users were not willing to pay more for tolerance. For Value/Cost, replication is rarely used due to its high relative resource cost. This results in a large proportion of tasks without fault tolerance, and hence, and large number of failures. For all heuristics, the number of replicated tasks decreases sharply as the queue length exceeds the size of the grid. This situation results from the high resource cost that replication requires; regardless of the heuristic employed, it is rare that a single task offers more value than two independent tasks.

Figure 7.3 (left) presents to total running times of all tasks in the queue. For queue lengths below the grid size, Value/Cost and Value/ERT deliver the fastest running times. This results from the inclusion of the cluster performances and failure probabilities in their utility functions. The strategies that do not observe these metrics, FCFS and Value, result in the longest running times. When the queue length exceeds the grid size, the Value heuristic achieves the lowest running time. This is a result of its preference for high value, and thus long tasks. By executing the long tasks early, the grid efficiency is kept high (rather than waiting for long tasks to finish after all others have finished). Note that in previous chapters, the value assigned to a task option was independent of its length; this reflected the fact that

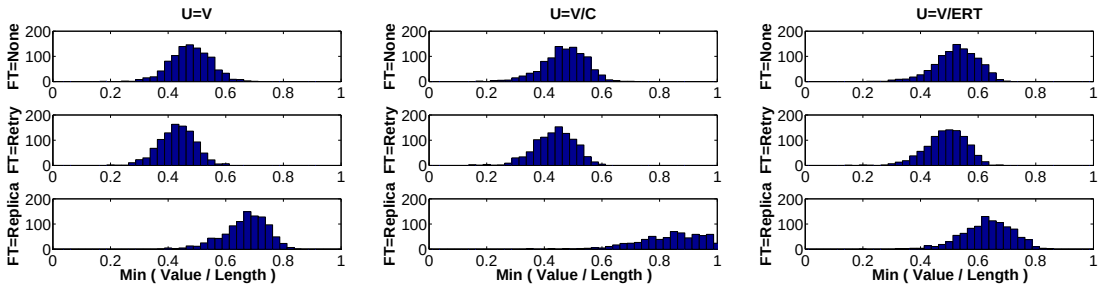

Figure 7.4: Minimum value of selected task options for all 1000 experiments and queue length 160

the value was an amount to be paid for a single allocation period, not the duration of the task. When length is decoupled from the value in this manner, the pure credit-value heuristics do not demonstrate an improvement in overall completion time. In this experiment, however, length is incorporated into the value because tasks cannot be preempted once they have been allocated resources; thus the value assigned to an option corresponds to the total amount paid for the entire duration of a task.

In figure 7.3 (right), we can observe the profit earned normalized to the total available profit. FCFS is consistently earning 50% of the available profit, as it randomly selects options. When the queue length is smaller than the grid size, Value finds the optimal profit in all cases. This situation changes when reallocation is required. In this case, the selected options are largely dictated by the resources that come available, not by their value. This effect is expected to be decreased if the time between placements is increased; if we wait for more resources to come available, we can make a better decision, however the utilization may drop. Techniques such as backfilling have been previous shown [54] to negate this effect.

Figure 7.4 shows histograms of the minimum value earned for each fault tolerance technique over all 1000 experiments. The plots correspond to a queue length of 160. From these plots we can infer the amount which must be offered to be selected for a desired fault tolerance technique. For the Value heuristic, retry is the cheapest option. Counterintuitively, users need to pay a premium for no fault tolerance; equivalently,

Table 7.3: Mean correlations between task value (V), value/length (V/L) and startup delay (D) for each heuristic at queue length 640

Corr.	FCFS	Value	Value/Cost	Value/ERT
V, D	$0 \pm .04$	$-.9283 \pm .0003$	$-.064 \pm .001$	$-.065 \pm .001$
V/L, D	$0 \pm .001$	$-.116 \pm .001$	$-.475 \pm .001$	$-.478 \pm .001$

they need to not offer more for fault tolerance if it is not desired. Replication is quite expensive, at nearly twice the price of retry. This is a result of the constraints which lead to competition for resources. When resource cost is used in the heuristic, as it is in Value/Cost, replication becomes very expensive. In this case, we see that in the rare cases that replication is selected, it had offered over 0.9 in normalized value. For the Value/ERT heuristic, we see pricing which is similar to Value. Value/ERT is desirable because it features desirable pricing characteristics while maintaining the timing properties of the schedule.

Finally, table 7.3 presents the mean correlations between startup delay, defined as the start time minus the submitted time, and both the maximum offered value and the maximum offered value relative to the estimated length. A negative correlation in this table implies that a larger offered value would result in a smaller startup delay. As expected, there is zero correlation shown for the reference strategy FCFS. The Value heuristic prefers tasks with the largest value; this is shown with a correlation between value and delay of close to -1. Value/ERT and Value/Cost do not prefer large absolute values. Instead, they prefer tasks with a large value relative to the estimated length. This is made clear in the second correlation in table 7.3, with both heuristics showing correlations of nearly -0.5. The correlation is weaker in these cases because Value/Cost and Value/ERT include metrics in addition to value in their utilities. In summary, the correlations presented show that the heuristics are correctly preferring highly valued tasks, thereby allowing users to communicate their task priorities to the grid.

7.6 Discussion and Conclusions

By reflecting on the results shown in the previous section, we can draw some general conclusions about the effectiveness of the knapsack formulation in matching tasks to fault tolerance techniques. First, we can conclude that all of the heuristics out perform FCFS in all metrics. Without considering the offered value, cluster performance, and fault probabilities the reference strategy cannot compete. Second, the best heuristic to employ depends on the goals of the system designers. The Value/Cost heuristic is interesting in that it emphasizes the cost of performing replication, however it does not perform better than the other heuristics in any of the metrics. When resources are abundant, Value will always optimize profit while having a poor running time. The Value/ERT heuristic presents the compromise solution. By sacrificing a small amount of profit, it is able to decrease the number of failures and the total running time.

In congested grids, it can be concluded that replication is rarely economically feasible, regardless of the allocation strategy employed. It is illogical for users to pay more of a premium than the fault probability, except in the rare cases of extremely urgent tasks. Given this, it is notable that all of the heuristics will still replicate tasks if they are offered enough. This provides users the ability to intervene and prioritize tasks that are critically important.

In summary, this chapter has presented a solution to the problem of selecting an optimal task placement to resources and mapping to available fault tolerance techniques. By employing the combined utility/knapsack solution technique, we find the placements by optimizing the summed utility of all tasks. Using heuristics which take into account the offered value, the resource costs, and the estimated response time of an option, the formulation can be used to produce allocations that decrease failures while decreasing running time and increasing profit.

Chapter 8

Conclusions

This dissertation has presented a novel approach to the allocation of grid resources. Being motivated by work to develop a pan-Canadian computational grid, the approach presented applies two techniques to improve upon existing grid resource allocation strategies. Using a utility model, the relative merits of each allocation option for a given task can be quantified. A number of heuristics are introduced to implement quality-of-service policies on the grid. By formulating a knapsack problem, a nearly optimal allocation set can be found quickly and accurately.

8.1 Summary of Contributions

The key contributions of each chapter are outlined below.

Chapter 3: GridX1, A Pan-Canadian Computational Grid Chapter 3 presented an overview of the design of GridX1. First, we presented a GRAM interface to Condor-G, thereby providing access to a metascheduling service which is identical to the grid clusters. Next, we described scheduling policies and their implementations using the Condor matchmaking mechanism; policies include an round robin distribution, an estimated wait time algorithm, and a data location-based ranking system. Finally, we presented an interface which provides transparent access to all of the GridX1 resources. This was used to federate the shared, and otherwise unavailable,

resources into the LHC Compute Grid during the ATLAS data challenges.

Chapter 4: Grid Resource Allocation using a Utility Model and Knapsack Formulation Chapter 4 introduced the utility/knapsack approach to grid resource allocation. In this chapter, we introduced a utility model for grid resource allocation option valuation, and utility function heuristics for implementing various quality of service (QoS) policies. The policies presented use QoS metrics including a user assigned credit-value, estimated response time, and nearness to completion time. Next, we formulated the grid resource allocation problem for processors as a variant of the 0-1 multichoice multidimensional knapsack problem. Simulation results of the utility/knapsack approach and a number of reference strategies were presented. The utility/knapsack approach is shown to improve efficiency while allowing users to intervene with high priority tasks.

Chapter 5: Design and Analysis of the Utility/Knapsack Scheduling System Chapter 5 presented methodologies and results for the design and analysis of utility/knapsack-based scheduling systems. First, we presented sensitivity analyses of the utility function heuristics. Specifically, the weights of the credit-value and nearness to completion time metrics, and the shape of a sigmoidal utility function are varied to measure their effects on the resultant task placements. Next, the Plackett-Burman design technique was used to gain further insight into the utility heuristics while requiring a limited number of experiments. Both techniques are shown to result in significant revelations in the function of utility/knapsack scheduler, and are therefore recommended to be used during practical deployment of the technique.

Chapter 6: Allocation of Multiple Resource Types Chapter 6 presents an extension of the utility/knapsack approach to the allocation of multiple resource types. It describes a formulation of the allocation problem as a 0-1 MMKP and introduces quality of service policies and utility function heuristics which consider processor,

storage, and network metrics. Simulation results show that the utility/knapsack approach provides improved efficiency, decreased data transfer times, and still allow users to intervene with urgent tasks.

Chapter 7: Selection between Fault Tolerance Options using the Utility/Knapsack Approach In chapter 7, we use the utility/knapsack approach to select between fault tolerance techniques. We first present a utility model for evaluating the relative merits of varied fault tolerance techniques, namely retry and replication. Next, we formulate the allocation problem using the single resource type utility/knapsack approach. Simulation results which show increased efficiency and decreased numbers of failures in comparison with reference strategies.

8.2 Future Work

The resource allocation system presented in this dissertation has proven, in simulation studies, to be competitive with, and in many cases, improve upon the strategies available in the literature. I propose a number of areas for continued work in this area.

8.2.1 Further Policy and Resource Exploration

One of the main benefits of the utility model presented in this dissertation is that it allows for the incorporation of a variety of QoS metrics into the calculation of the utility of a given allocation option. It is therefore possible to begin evaluating of a new set of allocation policies which improve the overall quality of the resource allocations. One particular area for exploration would be in power-, energy-, and temperature-awareness. For development in the latter case, it may be possible to explore a model which allocates tasks according to their temperature profile [81]. This model exploits the variations in task "hotness" to allow for the even distribution of tasks according to the amount of heat they will generate, thereby preventing some chasses or racks from overheating. When applied across a grid, approaches such as this will allow for

the fair distribution of power, energy, and temperature burdens across the constituent resources.

8.2.2 A Utility/Knapsack Approach to Scheduling Workflows

Another future application of the utility/knapsack approach is for the allocation of task workflows. Contrary to the independent serial tasks studied in this dissertation, task workflows reflect the multi-staged nature of complex grid applications. Workflows are represented by directed acyclic graphs, with each node representing a sub-task of the workflow. At each node in the workflow, a number of options exist for meeting the allocation requirements of that node. By applying the utility model to these options, it may be possible evaluate their relative utility, and the utility may be globally optimizable using a formulation of the knapsack problem.

8.2.3 Grid Workload Standardization

This dissertation presented simulation results for a variety of workloads, including those that were generated randomly according to a bimodal distribution of task lengths, and those drawn from the traces of a real system at San Diego Supercomputing Center. Using this wide variety of workloads made it possible to draw general conclusions about the performance of the utility/knapsack approach in comparison to a number of reference strategies. Moving forward, it would be desirable to work towards a standardized set of workloads that could be used by all researchers in the field.

Efforts have already begun in this area. For example, the Grid Workloads Archive [82] at TU Delft has gathered workloads from a variety of grids, including NorduGrid [39] and the LCG [38]. Studies need to be made to characterize these workloads, and they should be used to measure the performance of existing strategies.

8.2.4 Implementation of a Utility/Knapsack Scheduler

Perhaps the most important next step for this project is an implementation of the utility/knapsack approach. This could be achieved either through the extension of an existing grid scheduling system, such as the Gridbus broker [28], or through the development of a new system. This implementation will undoubtedly reveal new challenges and will provide a platform for further investigations into grid resource management.

Bibliography

[1] I. Foster and C. Kesselman, eds., *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, second ed., Nov. 2003.

[2] S. McKee, "The ATLAS computing model: status, plans and future possibilities," in *Proceedings of the Conference on Computational Physics 2006*, pp. 231–234, July 2006.

[3] D. G. Goodenough, H. Chen, L. Di, A. Guan, Y. Wei, A. Dyk, and G. Hobart, "Grid-enabled OGC Environment for EO Data and Services in Support of Canada's Forest Applications," in *IEEE International Geoscience and Remote Sensing Symposium*, July 2007.

[4] L. Childers, T. Disz, R. Olson, M. E. Papka, R. Stevens, and T. Udeshi, "Access grid: Immersive group-to-group collaborative visualization," in *Proceedings of the 4th International Immersive Projection Technology Workshop*, 2000.

[5] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations," *International J. Supercomputer Applications*, vol. 15, 2001.

[6] Globus Alliance, "The Globus Toolkit." [Online]. Available: `http://www.globus.org/`.

[7] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids," in *Proceedings of the 5th ACM Conference on Computer and Communications Security Conference*, pp. 83–92, 1998.

[8] "Public Key Infrastructure (X.509) Charter." [Online]. Available: `http://www.ietf.org/html.charters/pkix-charter.html`.

[9] J. Bester, I. Foster, C. Kesselman, J. Tedesco, and S. Tuecke, "GASS: a data movement and access service for wide area computing systems," in *IOPADS '99: Proceedings of the sixth workshop on I/O in parallel and distributed systems*, (New York, NY, USA), pp. 78–88, ACM, 1999.

[10] "GridFTP Protocol Specification," Tech. Rep. GFD.20, Mar. 2003.

[11] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems," in *Lecture Notes in Computer Science*, vol. 1459, p. 62, Jan. 1998.

[12] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing*, IEEE Press, Aug. 2001.

[13] W. Yeong, T. Howes, and S. Kille, "Lightweight directory access protocol," 1995.

[14] "W3c web services website." `http://www.w3.org/2002/ws/`.

[15] "Globus web services resource framework website." `http://www.globus.org/wsrf/`.

[16] M. P. McRae, "Approval of WSRF v1.2 as OASIS Standard." tc-announce@lists.oasis-open.org. 3 Apr 2006. Available: `http://lists.oasis-open.org/archives/tc-announce/200604/msg00000.html`.

[17] D. Feitelson and A. Weil, "Utilization and predictability in scheduling the ibm sp2 with backfilling," in *Parallel Processing Symposium*, 1998.

[18] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *J. ACM*, vol. 24, no. 2, pp. 280–289, 1977.

[19] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *J. Parallel Distrib. Comput.*, vol. 61, no. 6, pp. 810–837, 2001.

[20] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems," *Journal of Parallel and Distributed Computing*, vol. 59, pp. 671–680, Nov. 1999.

[21] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," in *Proceedings of the 9th Heterogeneous Computing Systems Workshop (HCW 2000)*, (Cancun, Mexico), IEEE CS Press, Los Alamitos, CA, USA, 2000.

[22] J. K. Ousterhout, "Scheduling techniques for concurrent systems," in *Proceedings of the 3rd International Conference on Distributed Computer Systems*, pp. 22–30, Oct. 1982.

[23] R. Raman, M. Livny, and M. Solomon, "Matchmaking: distributed resource management for high throughput computing," in *The Seventh International Symposium on High Performance Distributed Computing*, pp. 140–146, July 1998.

[24] "Condor batch system website." `http://www.cs.wisc.edu/condor/`.

[25] J. Frey, T. Tannenbaum, M. Livny, I. Foster, and S. Tuecke, "Condor-g: A computation management agent for multi-institutional grids," in *Proceedings of the Tenth International Symposium on High Performance Distributed Computing*, IEEE Press, Aug. 2001.

[26] R. Buyya, *Economic-based Distributed Resource Management and Scheduling for Grid Computing*. PhD thesis, Monash University, Melbourne, Australia, 2002.

[27] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/g: An architecture for a resource management and scheduling system in a global computational grid," in *The Fourth International Conference on High-Performance Computing in the Asia-Pacific Region*, 2000.

[28] R. Buyya and S. Venugopal, "The gridbus toolkit for service oriented grid and utility computing: an overview and status report," in *1st IEEE International Workshop on Grid Economics and Business Models*, pp. 19–66, Apr. 2004.

[29] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya, "Libra: A computational economy-based job scheduling system for clusters," *Software: Practice and Experience*, vol. 34, pp. 573–590, May 2004.

[30] C. S. Yeo and R. Buyya, "Integrated risk analysis for a commercial computing service," in *Proceedings of the 21st IEEE International Parallel and Distributed Processing Symposium (IPDPS 2007)*, Mar. 2007.

[31] S. Khan, K. F. Li, E. G. Manning, R. Watson, and G. C. Shoja, "Optimal quality of service routing and admission control using the utility model," *Future Generation Computer Systems*, vol. 19, pp. 1063–1073, Oct. 2003.

[32] G. Mounie, C. Rapine, and D. Trystram, "Efficient approximation algorithms for scheduling malleable tasks," in *Proceedings of the 11th annual ACM Symposium on Parallel Algorithms and Architecture*, pp. 23–32, 1999.

[33] R. Parra-Hernandez, *A Study of Knapsack-based Admission and Allocation Techniques*. PhD thesis, University of Victoria, Canada, 2005.

[34] R. Parra-Hernandez and N. Dimopoulos, "A New Heuristic for Solving the Multichoice Multidimensional Knapsack Problem," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 35, no. 5, pp. 708–717, 2005.

[35] A. Agarwal, M. Ahmed, A. Berman, B. Caron, A. Charbonneau, D. Deatrich, R. Desmarais, A. Dimopoulos, I. Gable, L. Groer, R. Haria, R. Impey, L. Klektau, C. Lindsay, G. Mateescu, Q. Matthews, A. Norton, W. Podaima, D. Quesnel, R. Simmonds, R. Sobie, B. S. Arnaud, C. Usher, D. Vanderster, M. Vetterli, R. Walker, and M. Yuen, "GridX1: A Canadian computational grid," *Future Generation Computer Systems*, vol. 23, pp. 680–687, June 2007.

[36] The TeraGrid Project, 2005. [Online]. Available: `http://www.teragrid.org/`.

[37] The Open Science Grid, 2006. [Online]. Available: http://www.opensciencegrid.org/.

[38] The LHC Computing Grid Project, 2005. [Online]. Available: http://lcg.web.cern.ch/LCG/.

[39] The NorduGrid Collaboration, 2005. [Online]. Available: http://www.nordugrid.org/.

[40] R. Alfieri, R. Barbera, P. Belluomo, A. Cavalli, R. Cecchini, A. Chierici, V. Ciaschini, L. Dell'Agnello, F. Donno, E. Ferro, A. Forte, L. Gaido, A. Ghiselli, A. Gianoli, A. Italiano, S. Lusso, M. Luvisetto, P. Mastroserio, M. Mazzucato, D. Mura, M. Reale, L. Salconi, G. Sava, M. Serra, F. Spataro, F. Taurino, G. Tortone, L. Vaccarossa, M. Verlato, and G. V. Finzi, "The INFN-Grid Testbed," *Future Generation Computer Systems*, vol. 21, pp. 249–258, 2005.

[41] T. Hey and A. E. Trefethen, "The UK e-Science Core Programme and the Grid," *Future Generation Computer Systems*, vol. 18, pp. 1017–1031, 2002.

[42] E. Seidel, G. Allen, A. Merzky, and J. Nabrzyski, "GridLab–a grid application toolkit and testbed," *Future Generation Computer Systems*, vol. 18, pp. 1143–1153, 2002.

[43] Virtual Data Toolkit, 2005. [Online]. Available: http://www.cs.wisc.edu/vdt/.

[44] The Enabling Grids for E-sciencE (EGEE) Project, 2006. [Online]. Available: http://www.eu-egee.org/.

[45] M. Ellart, M. Grønager, A. Konstantinov, B. Kónya, J. Lindemann, I. Livenson, J. Nielson, M. Niinimäki, O. Smirnova, and A. Wäänänen, "Advanced Resource Connector middleware for lightweight computational Grids," *Future Generation Computer Systems*, vol. 23, pp. 219–240, 2007.

[46] Portable Batch System Professional, 2005. [Online]. Available: http://www.altair.com/pbspro/.

[47] OpenPBS, 2005. [Online]. Available: http://www.openpbs.org/.

[48] J. Novotny, S. Tuecke, and V. Welch, "An online credential repository for the grid: Myproxy," IEEE Press, Aug. 2001.

[49] S. Smallen, C. Olschanowsky, K. Ericson, P. Beckman, and J. M. Schopf, "The Inca Test Harness and Reporting Framework," in *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, (Washington, DC, USA), p. 55, IEEE Computer Society, 2004.

[50] H. B. Newman, I. C. Legrand, P. Galvez, R. Voicu, and C. Cirstoiu, "MonALISA: A Distributed Monitoring Service Architecture," in *Conference on Computing and High Energy Physics (CHEP)*, 2003.

[51] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, vol. 30, pp. 817–840, July 2004.

[52] The ATLAS Data Challenge, 2005. [Online]. Available: `http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/DC/`.

[53] R. Walker, M. Vetterli, R. Impey, G. Mateescu, B. Caron, A. Agarwal, A. Dimopoulos, L. Klektau, C. Lindsay, R. J. Sobie, and D. Vanderster, "Federating Grids: LCG Meets Canadian HEPGrid," in *Proceedings of Computing in High Energy and Nuclear Physics 2004*, 2004.

[54] R. Parra-Hernandez, D. Vanderster, and N. J. Dimopoulos, "Resource Management and Knapsack Formulations on the Grid," in *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, (Washington, DC, USA), pp. 94–101, IEEE Computer Society, 2004.

[55] D. C. Vanderster, N. J. Dimopoulos, R. Parra-Hernandez, and R. J. Sobie, "Evaluation of Knapsack-Based Scheduling Using the NPACI JOBLOG," in *Proceedings of the 20th International Symposium on High-Performance Computing in an Advanced Collaborative Environment*, (Washington, DC, USA), p. 15, IEEE Computer Society, 2006.

[56] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson, "Economic models for management of resources in peer-to-poeer and grid computing," in *Proceedings of the International Conference on Commercial Applications for High-Performance Computing*, Aug. 2001.

[57] S. Venugopal and R. Buyya, "A Deadline and Budget Constrained Scheduling Algorithm for e-Science Applications on Data Grids," in *Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2005)*, vol. 3719 of *Lecture Notes in Computer Science*, (Melbourne, Australia.), Springer-Verlag, Berlin, Germany, Oct. 2005.

[58] J. Gomoluch and M. Schroeder, "Market-based resource allocation for grid computing," in *Middleware 2003 Workshops: Rio de Janeiro, Brazil*, pp. 211–218, June 2003.

[59] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proceedings of the 19th ACM Symposium on Operating Systems Principles*, Oct. 2003.

[60] R. Parra-Hernandez and N. Dimopoulos, "Heuristic Approaches for Solving the Multidimensional Knapsack Problem (MKP)," *WSEAS Transactions on Systems*, pp. 248–253, Apr. 2002.

[61] H. Pirkul, "A Heuristic Solution Procedure for the Multiconstraint Zero-One Knapsack Problem," *Naval Research Logistics*, pp. 161–172, 1987.

[62] R. Loulou and E. Michaelides, "New greedy-like heuristics for the multidimensional 0-1 knapsack problem," *Operations Research*, pp. 1101–1114, Nov. 1979.

[63] NPACI JOBLOG Job Trace Repository, 2000. [Online]. Available: `http://joblog.npaci.edu/`.

[64] H. Casanova, "Simgrid: A Toolkit for the Simulation of Application Scheduling," in *Proceedings of the 1st International Symposium on Cluster Computing and the Grid*, (Washington, DC, USA), pp. 430–437, IEEE Computer Society, 2001.

[65] J. L. Devore, *Probability and Statistics for Engineering and the Sciences*. Thomson Brooks/Cole, fourth ed., 1995.

[66] D. C. Vanderster and N. J. Dimopoulos, "Sensitivity analysis of knapsack-based task scheduling on the grid," in *Proceedings of the 20th Annual International Conference on Supercomputing*, (New York, NY, USA), pp. 317–323, ACM, 2006.

[67] D. C. Vanderster, N. J. Dimopoulos, and R. J. Sobie, "Improved Grid Metascheduler Design using the Plackett-Burman Methodology," in *21st International Symposium on High Performance Computing Systems and Applications*, p. 9, May 2007.

[68] R. Plackett and J. Burman, "The design of optimum multifactorial experiments," *Biometrika*, vol. 33, pp. 305–325, 6 1946.

[69] J. J. Yi, D. J. Lilja, and D. M. Hawkins, "Improving computer architecture simulation methodology by adding statistical rigor," *IEEE Transactions on Computers*, vol. 54, pp. 1360–1373, 11 2005.

[70] D. Montgomery, *Design and Analysis of Experiments*. Wiley, fifth ed., 2001.

[71] D. C. Vanderster, N. J. Dimopoulos, and R. J. Sobie, "Metascheduling Multiple Resource Types using the MMKP," in *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pp. 231–237, Sept. 2006.

[72] D. C. Vanderster, N. J. Dimopoulos, and R. J. Sobie, "Intelligent Selection of Fault Tolerance Techniques on the Grid," in *Proceedings of the 3rd IEEE Conference on e-Science and Grid Computing*, Dec. 2007.

[73] R. Medeiros, W. Cirne, F. Brasileiro, and J. Sauvé, "Faults in Grids: Why are they so bad and What can be done about it?," in *Proceedings of the Fourth International Workshop on Grid Computing*, (Washington, DC, USA), p. 18, IEEE Computer Society, 2003.

[74] J. Yu and R. Buyya, "A Taxonomy of Workflow Management Systems for Grid Computing," *Journal of Grid Computing*, vol. 3, pp. 171–200, 2006.

[75] J. H. Abawajy, "Fault-Tolerant Scheduling Policy for Grid Computing Systems," in *18th International Parallel and Distributed Processing Symposium*, (Los Alamitos, CA, USA), pp. 238–244, IEEE Computer Society, 2004.

[76] C. Anglano and M. Canonico, "Fault-Tolerant Scheduling for Bag-of-Tasks Grid Applications," in *Advances in Grid Computing - EGC 2005*, pp. 630–639, 2005.

[77] K. Limaye, B. Leangsuksun, Y. Liu, Z. Greenwood, S. L. Scott, R. Libby, and K. Chanchio, "Reliability-Aware Resource Management for Computational Grid/Cluster Environments," in *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing*, (Washington, DC, USA), pp. 211–218, IEEE Computer Society, 2005.

[78] D. Nurmi, J. Brevik, and R. Wolski, "Modeling Machine Availability in Enterprise and Wide-Area Distributed Computing Environments," in *Euro-Par 2005 Parallel Processing*, pp. 432–441.

[79] R. Wolski, N. T. Spring, and J. Hayes, "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Future Generation Computer Systems*, vol. 15, no. 5-6, pp. 757–768, 1999.

[80] R. Wolski, "Experiences with Predicting Resource Performance On-line in Computational Grid Settings," *SIGMETRICS Perform. Eval. Rev.*, vol. 30, no. 4, pp. 41–49, 2003.

[81] D. Vanderster, A. Baniasadi, and N. Dimopoulos, "Exploiting task temperature profiling in temperature-aware task scheduling for computational clusters," in *The Twelfth Asia-Pacific Computer Systems Architecture Conference (ACSAC 2007)*, pp. 175–185, 2007.

[82] The Grid Workloads Archive, 2008. [Online]. Available: `http://gwa.ewi.tudelft.nl/`.

Appendix A

Full Data Tables

A.1 Data tables for chapter 4

The complete data tables for chapter 4 are included here for reference purposes. The tables corresponding to the random workload are in tables A.1 and A.2; the tables corresponding to the NPACI workload are in tables A.3 and A.4.

A.2 Data tables for chapter 6

The complete data tables for chapter 6 are included here in tables A.5 through A.7 for reference purposes.

Table A.1: Chapter 4: Random workload, overall completion time (s $*10^7$)

Reference

λ	FCFS	BF	m-m	Gang (τ)		
				6h	9h	30h
0.00	1.60 ± 0.02	1.37 ± 0.01	1.39 ± 0.01	1.93 ± 0.04	1.98 ± 0.04	2.35 ± 0.05
0.64	1.61 ± 0.01	1.37 ± 0.01	1.40 ± 0.01	1.67 ± 0.02	1.72 ± 0.03	2.20 ± 0.03
1.60	1.61 ± 0.02	1.39 ± 0.01	1.43 ± 0.01	1.52 ± 0.01	1.55 ± 0.01	2.03 ± 0.02
1.92	1.62 ± 0.02	1.48 ± 0.01	1.48 ± 0.01	1.52 ± 0.01	1.55 ± 0.01	1.98 ± 0.01
6.40	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.71 ± 0.03	4.71 ± 0.03	4.71 ± 0.03

Policy 1

λ	UM (τ)			UM+BF (τ)		
	6h	9h	30h	6h	9h	30h
0.00	1.44 ± 0.01	1.50 ± 0.01	1.98 ± 0.01	1.34 ± 0.01	1.34 ± 0.01	1.35 ± 0.01
0.64	1.44 ± 0.01	1.50 ± 0.01	1.97 ± 0.01	1.34 ± 0.01	1.35 ± 0.01	1.35 ± 0.01
1.60	1.44 ± 0.01	1.50 ± 0.01	1.93 ± 0.01	1.36 ± 0.01	1.36 ± 0.01	1.37 ± 0.01
1.92	1.48 ± 0.01	1.50 ± 0.01	1.90 ± 0.01	1.47 ± 0.01	1.47 ± 0.01	1.47 ± 0.01
6.40	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03

Policy 2

0.00	1.45 ± 0.01	1.52 ± 0.01	2.04 ± 0.01	1.34 ± 0.01	1.35 ± 0.01	1.35 ± 0.01
0.64	1.46 ± 0.01	1.52 ± 0.01	2.01 ± 0.01	1.35 ± 0.01	1.35 ± 0.01	1.35 ± 0.01
1.60	1.46 ± 0.01	1.51 ± 0.01	1.97 ± 0.02	1.36 ± 0.01	1.36 ± 0.01	1.36 ± 0.01
1.92	1.48 ± 0.01	1.51 ± 0.01	1.91 ± 0.01	1.47 ± 0.01	1.47 ± 0.01	1.47 ± 0.01
6.40	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03

Policy 3

0.00	1.39 ± 0.01	1.42 ± 0.01	1.63 ± 0.01	1.34 ± 0.01	1.34 ± 0.01	1.35 ± 0.01
0.64	1.40 ± 0.01	1.43 ± 0.01	1.63 ± 0.01	1.34 ± 0.01	1.35 ± 0.01	1.35 ± 0.01
1.60	1.41 ± 0.01	1.44 ± 0.01	1.64 ± 0.01	1.36 ± 0.01	1.36 ± 0.01	1.37 ± 0.01
1.92	1.47 ± 0.01	1.48 ± 0.01	1.65 ± 0.01	1.47 ± 0.01	1.47 ± 0.01	1.47 ± 0.01
6.40	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03

Policy 4

0.00	1.45 ± 0.01	1.48 ± 0.01	1.63 ± 0.01	1.37 ± 0.01	1.37 ± 0.01	1.37 ± 0.01
0.64	1.45 ± 0.01	1.49 ± 0.01	1.67 ± 0.01	1.37 ± 0.01	1.37 ± 0.01	1.37 ± 0.01
1.60	1.44 ± 0.01	1.48 ± 0.01	1.69 ± 0.01	1.36 ± 0.01	1.36 ± 0.01	1.37 ± 0.01
1.92	1.48 ± 0.01	1.50 ± 0.01	1.70 ± 0.01	1.47 ± 0.01	1.47 ± 0.01	1.47 ± 0.01
6.40	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03

Policy 5

0.00	1.41 ± 0.01	1.44 ± 0.01	1.58 ± 0.01	1.36 ± 0.01	1.36 ± 0.01	1.37 ± 0.01
0.64	1.41 ± 0.01	1.44 ± 0.01	1.60 ± 0.01	1.36 ± 0.01	1.36 ± 0.01	1.37 ± 0.01
1.60	1.42 ± 0.01	1.44 ± 0.01	1.61 ± 0.01	1.36 ± 0.01	1.37 ± 0.01	1.37 ± 0.01
1.92	1.48 ± 0.01	1.49 ± 0.01	1.64 ± 0.01	1.47 ± 0.01	1.47 ± 0.01	1.47 ± 0.01
6.40	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03	4.70 ± 0.03

Table A.2: Chapter 4: Random workload, correlation of credit-value-metric with speedup relative to FCFS.

Reference

λ	FCFS	BF	m-m	Gang (τ)		
				6h	9h	30h
0.00	–	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.64	–	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01
1.60	–	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01
1.92	–	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01
6.40	–	0.00 ± 0.01	0.00 ± 0.01	0.01 ± 0.01	0.01 ± 0.01	0.01 ± 0.01

Policy 1

λ	UM (τ)			UM+BF (τ)		
	6h	9h	30h	6h	9h	30h
0.00	0.24 ± 0.01	0.25 ± 0.01	0.30 ± 0.01	0.23 ± 0.01	0.24 ± 0.01	0.26 ± 0.01
0.64	0.16 ± 0.01	0.16 ± 0.01	0.17 ± 0.01	0.13 ± 0.02	0.15 ± 0.02	0.15 ± 0.02
1.60	0.07 ± 0.01	0.07 ± 0.01	0.11 ± 0.01	0.06 ± 0.01	0.05 ± 0.01	0.08 ± 0.01
1.92	0.03 ± 0.01	0.03 ± 0.01	0.09 ± 0.01	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.01
6.40	0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01

Policy 2

0.00	0.10 ± 0.01	0.11 ± 0.01	0.14 ± 0.01	0.10 ± 0.01	0.11 ± 0.01	0.13 ± 0.01
0.64	0.10 ± 0.01	0.12 ± 0.01	0.14 ± 0.01	0.09 ± 0.01	0.11 ± 0.01	0.08 ± 0.01
1.60	0.07 ± 0.01	0.07 ± 0.01	0.13 ± 0.01	0.06 ± 0.01	0.05 ± 0.01	0.05 ± 0.01
1.92	0.01 ± 0.01	0.03 ± 0.01	0.13 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.01 ± 0.01
6.40	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01

Policy 3

0.00	0.12 ± 0.01	0.13 ± 0.01	0.16 ± 0.01	0.12 ± 0.01	0.13 ± 0.01	0.16 ± 0.01
0.64	0.11 ± 0.01	0.12 ± 0.01	0.14 ± 0.01	0.09 ± 0.02	0.10 ± 0.02	0.09 ± 0.02
1.60	0.08 ± 0.01	0.07 ± 0.01	0.12 ± 0.01	0.06 ± 0.01	0.04 ± 0.01	0.04 ± 0.01
1.92	0.02 ± 0.01	0.03 ± 0.01	0.10 ± 0.01	0.01 ± 0.01	0.01 ± 0.01	0.01 ± 0.01
6.40	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.01	0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01

Policy 4

0.00	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.64	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
1.60	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
1.92	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01
6.40	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01

Policy 5

0.00	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.64	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01
1.60	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
1.92	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
6.40	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.01	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.01

Table A.3: Chapter 4: NPACI workload, utility-based strategies: overall completion time (s $*10^6$).

Reference

k_λ	FCFS	BF	m-m	Gang (τ)		
				6h	9h	30h
0.0	1.9 ± 0.5	1.7 ± 0.4	1.7 ± 0.4	5.6 ± 1.1	7.5 ± 1.4	21.8 ± 3.8
0.1	1.9 ± 0.5	1.7 ± 0.4	1.8 ± 0.4	5.7 ± 1.1	7.5 ± 1.4	21.8 ± 3.8
0.2	1.9 ± 0.5	1.7 ± 0.4	1.8 ± 0.4	5.7 ± 1.1	7.5 ± 1.4	21.8 ± 3.8
0.5	2.3 ± 0.5	2.2 ± 0.4	2.2 ± 0.5	5.5 ± 1.0	7.5 ± 1.4	21.8 ± 3.8
1.0	4.2 ± 0.8	4.2 ± 0.8	4.2 ± 0.8	5.4 ± 1.0	7.3 ± 1.3	21.7 ± 3.8

Policy 1

k_λ	UM (τ)			UM+BF (τ)		
	6h	9h	30h	6h	9h	30h
0.0	5.4 ± 1.0	7.4 ± 1.4	22.3 ± 4.0	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.1	5.4 ± 1.0	7.4 ± 1.4	22.3 ± 4.0	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.2	5.4 ± 1.0	7.4 ± 1.4	22.3 ± 4.0	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.5	5.4 ± 1.0	7.3 ± 1.4	22.3 ± 4.0	2.2 ± 0.5	2.2 ± 0.5	2.2 ± 0.4
1.0	5.4 ± 1.0	7.3 ± 1.3	22.2 ± 4.0	4.2 ± 0.8	4.2 ± 0.8	4.2 ± 0.8

Policy 2

0.0	5.6 ± 1.1	7.7 ± 1.4	22.7 ± 4.1	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.1	5.6 ± 1.1	7.6 ± 1.4	22.7 ± 4.1	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.2	5.6 ± 1.1	7.7 ± 1.4	22.7 ± 4.1	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.5	5.5 ± 1.1	7.6 ± 1.4	22.7 ± 4.1	2.2 ± 0.5	2.2 ± 0.5	2.2 ± 0.4
1.0	5.4 ± 1.0	7.4 ± 1.4	22.7 ± 4.1	4.2 ± 0.8	4.2 ± 0.8	4.2 ± 0.8

Policy 3

0.0	4.4 ± 0.9	5.9 ± 1.1	16.8 ± 2.9	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.1	4.5 ± 0.9	6.0 ± 1.1	16.9 ± 2.9	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.2	4.6 ± 0.9	6.1 ± 1.1	17.0 ± 2.9	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.5	4.6 ± 0.9	6.2 ± 1.1	17.3 ± 3.0	2.2 ± 0.5	2.2 ± 0.5	2.2 ± 0.5
1.0	4.8 ± 0.9	6.1 ± 1.1	17.6 ± 3.0	4.2 ± 0.8	4.2 ± 0.8	4.2 ± 0.8

Policy 4

0.0	4.1 ± 0.8	5.2 ± 1.0	13.3 ± 2.2	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.1	4.1 ± 0.8	5.2 ± 1.0	13.3 ± 2.2	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.2	4.1 ± 0.8	5.2 ± 1.0	13.3 ± 2.2	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.5	4.1 ± 0.8	5.2 ± 1.0	13.3 ± 2.2	2.2 ± 0.5	2.2 ± 0.4	2.2 ± 0.4
1.0	4.6 ± 0.8	5.4 ± 1.0	13.3 ± 2.2	4.2 ± 0.8	4.2 ± 0.8	4.2 ± 0.8

Policy 5

0.0	4.0 ± 0.8	5.1 ± 0.9	13.3 ± 2.2	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.1	4.0 ± 0.8	5.1 ± 1.0	13.3 ± 2.2	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.2	4.0 ± 0.8	5.1 ± 0.9	13.3 ± 2.2	1.7 ± 0.4	1.7 ± 0.4	1.7 ± 0.4
0.5	4.1 ± 0.8	5.2 ± 1.0	13.3 ± 2.2	2.2 ± 0.5	2.2 ± 0.5	2.2 ± 0.4
1.0	4.5 ± 0.8	5.4 ± 1.0	13.3 ± 2.2	4.2 ± 0.8	4.2 ± 0.8	4.2 ± 0.8

Table A.4: Chapter 4: NPACI workload, correlation of credit-value-metric with speedup relative to FCFS.

Reference

k_λ	FCFS	BF	m-m	Gang (τ)		
				6h	9h	30h
0.0	–	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01
0.1	–	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01
0.2	–	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01
0.5	–	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01
1.0	–	0.00 ± 0.01	0.00 ± 0.01	0.01 ± 0.01	0.01 ± 0.01	0.00 ± 0.01

Policy 1

k_λ	UM (τ)			UM+BF (τ)		
	6h	9h	30h	6h	9h	30h
0.0	0.39 ± 0.04	0.39 ± 0.04	0.39 ± 0.04	0.10 ± 0.04	0.07 ± 0.04	0.02 ± 0.02
0.1	0.13 ± 0.05	0.14 ± 0.05	0.25 ± 0.05	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.2	0.10 ± 0.03	0.09 ± 0.03	0.17 ± 0.05	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.5	0.09 ± 0.04	0.09 ± 0.04	0.08 ± 0.02	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01
1.0	0.07 ± 0.03	0.09 ± 0.03	0.10 ± 0.03	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01

Policy 2

0.0	0.17 ± 0.04	0.17 ± 0.04	0.17 ± 0.04	0.04 ± 0.01	0.03 ± 0.01	0.01 ± 0.01
0.1	0.13 ± 0.03	0.14 ± 0.04	0.14 ± 0.04	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.2	0.08 ± 0.01	0.11 ± 0.03	0.13 ± 0.04	0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.5	0.05 ± 0.02	0.05 ± 0.02	0.06 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
1.0	0.04 ± 0.02	0.06 ± 0.02	0.07 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01

Policy 3

0.0	0.18 ± 0.03	0.18 ± 0.03	0.18 ± 0.03	0.05 ± 0.02	0.04 ± 0.02	0.01 ± 0.01
0.1	0.13 ± 0.03	0.14 ± 0.04	0.15 ± 0.04	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.2	0.07 ± 0.01	0.11 ± 0.03	0.14 ± 0.04	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.5	0.05 ± 0.02	0.05 ± 0.02	0.06 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01
1.0	0.04 ± 0.02	0.06 ± 0.02	0.06 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01

Policy 4

0.0	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.1	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.2	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01
0.5	0.00 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
1.0	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01

Policy 5

0.0	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01
0.1	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01
0.2	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01
0.5	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01
1.0	0.00 ± 0.01	0.00 ± 0.01	0.00 ± 0.01	-0.01 ± 0.01	0.00 ± 0.01	0.00 ± 0.01

Table A.5: Chapter 6: Overall Completion Time Normalized to Basic FCFS

d	Data FCFS	UM Data	UM NonData	UM Data+Money	UM NonData+Money
1	1.01 ± 0.01	0.88 ± 0.06	0.91 ± 0.06	0.88 ± 0.06	0.89 ± 0.05
2	0.98 ± 0.01	0.87 ± 0.05	0.91 ± 0.06	0.87 ± 0.05	0.89 ± 0.05
3	1.00 ± 0.01	0.88 ± 0.06	0.95 ± 0.06	0.88 ± 0.05	0.93 ± 0.05
4	1.00 ± 0.01	0.88 ± 0.05	0.96 ± 0.06	0.87 ± 0.05	0.94 ± 0.05
5	1.00 ± 0.01	0.90 ± 0.05	1.02 ± 0.04	0.88 ± 0.05	0.97 ± 0.06
6	1.00 ± 0.01	0.88 ± 0.05	1.03 ± 0.05	0.89 ± 0.04	0.99 ± 0.06
7	1.01 ± 0.01	0.89 ± 0.05	1.04 ± 0.06	0.88 ± 0.04	1.03 ± 0.07
8	0.99 ± 0.01	0.88 ± 0.05	1.06 ± 0.04	0.89 ± 0.04	1.02 ± 0.06
9	1.00 ± 0.01	0.89 ± 0.05	1.06 ± 0.06	0.88 ± 0.04	1.04 ± 0.06
10	1.00 ± 0.01	0.89 ± 0.04	1.17 ± 0.04	0.90 ± 0.03	1.06 ± 0.07
20	0.94 ± 0.05	0.92 ± 0.03	1.60 ± 0.32	0.93 ± 0.03	1.23 ± 0.09
30	1.04 ± 0.03	1.11 ± 0.15	1.59 ± 0.20	1.05 ± 0.06	1.51 ± 0.18
40	0.99 ± 0.02	1.11 ± 0.15	1.82 ± 0.27	1.09 ± 0.13	1.76 ± 0.28
50	1.00 ± 0.02	1.03 ± 0.11	1.90 ± 0.21	1.03 ± 0.10	1.82 ± 0.26
60	0.99 ± 0.02	0.99 ± 0.11	1.92 ± 0.22	1.01 ± 0.08	1.99 ± 0.29
70	0.99 ± 0.02	0.94 ± 0.08	1.99 ± 0.18	0.97 ± 0.06	2.15 ± 0.33
80	0.97 ± 0.01	0.91 ± 0.07	2.15 ± 0.22	0.94 ± 0.05	2.18 ± 0.26
90	0.99 ± 0.01	0.93 ± 0.06	2.53 ± 0.31	0.95 ± 0.03	2.37 ± 0.32
100	0.99 ± 0.02	0.90 ± 0.05	2.89 ± 0.48	0.94 ± 0.03	2.44 ± 0.31
125	0.99 ± 0.01	0.91 ± 0.04	2.51 ± 0.39	0.92 ± 0.02	2.54 ± 0.34
150	1.00 ± 0.01	0.93 ± 0.03	2.69 ± 0.44	0.91 ± 0.02	2.64 ± 0.40
175	1.01 ± 0.01	0.95 ± 0.02	2.84 ± 0.47	0.93 ± 0.02	2.56 ± 0.39
200	1.02 ± 0.01	0.94 ± 0.03	2.58 ± 0.50	0.92 ± 0.01	2.57 ± 0.40
225	1.02 ± 0.01	0.94 ± 0.03	2.68 ± 0.48	0.91 ± 0.02	2.79 ± 0.41
250	1.02 ± 0.01	0.96 ± 0.02	2.46 ± 0.38	0.92 ± 0.02	2.48 ± 0.36
275	1.02 ± 0.01	0.95 ± 0.02	2.36 ± 0.38	0.91 ± 0.02	2.48 ± 0.37
300	1.02 ± 0.01	0.94 ± 0.01	2.34 ± 0.33	0.91 ± 0.01	2.34 ± 0.29
325	1.02 ± 0.01	0.93 ± 0.02	2.48 ± 0.32	0.90 ± 0.02	2.28 ± 0.32
350	1.02 ± 0.01	0.93 ± 0.01	2.29 ± 0.43	0.90 ± 0.02	2.23 ± 0.35
375	1.02 ± 0.01	0.93 ± 0.02	2.26 ± 0.41	0.89 ± 0.01	2.24 ± 0.40
400	1.02 ± 0.01	0.94 ± 0.03	2.21 ± 0.35	0.91 ± 0.02	2.18 ± 0.40
425	1.04 ± 0.01	0.94 ± 0.02	1.99 ± 0.25	0.92 ± 0.02	1.89 ± 0.20
450	1.02 ± 0.01	0.96 ± 0.02	2.09 ± 0.51	0.93 ± 0.04	1.69 ± 0.12
475	1.02 ± 0.01	0.96 ± 0.02	1.69 ± 0.23	0.92 ± 0.04	1.52 ± 0.10
500	1.02 ± 0.01	0.96 ± 0.02	1.47 ± 0.09	0.94 ± 0.04	1.38 ± 0.07
550	1.03 ± 0.01	0.98 ± 0.02	1.31 ± 0.06	0.97 ± 0.04	1.20 ± 0.05
600	1.03 ± 0.01	0.97 ± 0.04	1.07 ± 0.03	0.91 ± 0.02	1.01 ± 0.02
650	1.03 ± 0.01	0.97 ± 0.03	1.02 ± 0.03	0.91 ± 0.02	0.97 ± 0.02
700	1.03 ± 0.01	0.97 ± 0.02	0.98 ± 0.03	0.91 ± 0.02	0.96 ± 0.02
750	1.03 ± 0.01	0.95 ± 0.02	0.95 ± 0.03	0.91 ± 0.02	0.94 ± 0.03
800	1.02 ± 0.01	0.95 ± 0.02	0.96 ± 0.02	0.91 ± 0.01	0.94 ± 0.01
850	1.02 ± 0.01	0.96 ± 0.02	0.96 ± 0.02	0.93 ± 0.01	0.94 ± 0.01
900	1.02 ± 0.01	0.96 ± 0.01	0.96 ± 0.02	0.94 ± 0.01	0.95 ± 0.01
950	1.02 ± 0.01	0.96 ± 0.01	0.97 ± 0.02	0.94 ± 0.01	0.95 ± 0.01
1000	1.02 ± 0.01	0.96 ± 0.01	0.96 ± 0.01	0.94 ± 0.01	0.95 ± 0.01
Mean	1.01 ± 0.01	0.94 ± 0.04	1.68 ± 0.20	0.92 ± 0.03	1.62 ± 0.18

Table A.6: Chapter 6: Total Transfer Time Normalized to Basic FCFS

d	Data FCFS	UM Data	UM NonData	UM Data+Money	UM NonData+Money
1	0.92 ± 0.03	1.02 ± 0.05	2.34 ± 0.50	1.04 ± 0.05	2.33 ± 0.55
2	0.86 ± 0.05	0.97 ± 0.07	2.19 ± 0.44	0.99 ± 0.06	2.25 ± 0.50
3	0.94 ± 0.04	1.01 ± 0.08	2.34 ± 0.48	1.05 ± 0.08	2.40 ± 0.51
4	0.89 ± 0.05	1.00 ± 0.08	2.26 ± 0.45	1.02 ± 0.06	2.33 ± 0.47
5	0.94 ± 0.04	1.04 ± 0.09	2.36 ± 0.45	1.06 ± 0.09	2.33 ± 0.44
6	0.91 ± 0.04	1.01 ± 0.07	2.32 ± 0.44	1.02 ± 0.06	2.24 ± 0.37
7	0.95 ± 0.04	1.03 ± 0.09	2.28 ± 0.40	1.05 ± 0.09	2.26 ± 0.34
8	0.94 ± 0.04	1.03 ± 0.09	2.28 ± 0.40	1.05 ± 0.09	2.22 ± 0.35
9	0.95 ± 0.07	1.02 ± 0.09	2.26 ± 0.39	1.05 ± 0.10	2.12 ± 0.31
10	0.89 ± 0.04	0.98 ± 0.06	2.17 ± 0.41	0.99 ± 0.05	2.03 ± 0.33
20	0.91 ± 0.04	0.94 ± 0.05	2.21 ± 0.31	0.96 ± 0.03	1.91 ± 0.21
30	0.94 ± 0.04	0.94 ± 0.05	2.35 ± 0.25	0.97 ± 0.03	2.10 ± 0.21
40	0.89 ± 0.03	0.91 ± 0.04	2.37 ± 0.17	0.95 ± 0.02	2.29 ± 0.11
50	0.92 ± 0.04	0.92 ± 0.04	2.49 ± 0.18	0.99 ± 0.02	2.55 ± 0.13
60	0.90 ± 0.04	0.91 ± 0.04	2.68 ± 0.18	0.98 ± 0.01	2.78 ± 0.14
70	0.90 ± 0.05	0.89 ± 0.05	2.61 ± 0.14	0.98 ± 0.01	2.97 ± 0.19
80	0.89 ± 0.05	0.93 ± 0.04	3.02 ± 0.28	1.01 ± 0.02	3.19 ± 0.29
90	0.89 ± 0.04	0.92 ± 0.04	3.16 ± 0.36	0.99 ± 0.01	3.28 ± 0.35
100	0.90 ± 0.05	0.93 ± 0.04	3.30 ± 0.40	1.02 ± 0.01	3.46 ± 0.34
125	0.88 ± 0.05	0.96 ± 0.04	3.29 ± 0.47	1.01 ± 0.02	3.55 ± 0.35
150	0.87 ± 0.05	0.97 ± 0.04	3.33 ± 0.48	1.01 ± 0.01	3.59 ± 0.47
175	0.87 ± 0.05	1.00 ± 0.02	3.52 ± 0.56	1.04 ± 0.02	3.43 ± 0.49
200	0.87 ± 0.05	1.01 ± 0.02	3.27 ± 0.59	1.03 ± 0.01	3.35 ± 0.49
225	0.86 ± 0.05	1.01 ± 0.02	3.16 ± 0.52	1.02 ± 0.01	3.29 ± 0.44
250	0.86 ± 0.05	1.03 ± 0.01	3.01 ± 0.46	1.03 ± 0.01	3.08 ± 0.40
275	0.86 ± 0.05	1.02 ± 0.01	2.91 ± 0.48	1.03 ± 0.01	2.98 ± 0.34
300	0.86 ± 0.06	1.02 ± 0.01	2.80 ± 0.41	1.02 ± 0.01	2.84 ± 0.30
325	0.85 ± 0.06	1.01 ± 0.01	2.77 ± 0.40	1.02 ± 0.01	2.77 ± 0.30
350	0.86 ± 0.06	1.02 ± 0.02	2.70 ± 0.45	1.02 ± 0.01	2.70 ± 0.32
375	0.86 ± 0.06	1.02 ± 0.02	2.62 ± 0.43	1.02 ± 0.01	2.66 ± 0.36
400	0.86 ± 0.06	1.02 ± 0.02	2.51 ± 0.36	1.02 ± 0.01	2.57 ± 0.35
425	0.87 ± 0.06	1.02 ± 0.02	2.42 ± 0.35	1.02 ± 0.01	2.34 ± 0.25
450	0.87 ± 0.06	1.01 ± 0.02	2.46 ± 0.50	1.02 ± 0.01	2.21 ± 0.21
475	0.87 ± 0.06	1.00 ± 0.02	2.20 ± 0.35	1.00 ± 0.01	2.05 ± 0.19
500	0.86 ± 0.05	1.00 ± 0.02	1.95 ± 0.19	1.00 ± 0.01	1.89 ± 0.14
550	0.87 ± 0.05	1.00 ± 0.02	1.77 ± 0.16	1.02 ± 0.02	1.63 ± 0.08
600	0.87 ± 0.05	0.98 ± 0.02	1.48 ± 0.09	0.99 ± 0.01	1.48 ± 0.08
650	0.88 ± 0.04	0.99 ± 0.02	1.42 ± 0.10	0.99 ± 0.01	1.40 ± 0.07
700	0.89 ± 0.04	0.99 ± 0.01	1.37 ± 0.10	0.99 ± 0.01	1.36 ± 0.07
750	0.90 ± 0.03	0.99 ± 0.02	1.29 ± 0.09	0.98 ± 0.01	1.31 ± 0.08
800	0.91 ± 0.03	0.99 ± 0.02	1.27 ± 0.09	0.99 ± 0.01	1.26 ± 0.07
850	0.92 ± 0.02	0.99 ± 0.02	1.24 ± 0.08	0.99 ± 0.01	1.21 ± 0.06
900	0.93 ± 0.02	1.00 ± 0.02	1.21 ± 0.07	1.00 ± 0.01	1.18 ± 0.06
950	0.94 ± 0.02	1.00 ± 0.02	1.18 ± 0.07	1.00 ± 0.01	1.18 ± 0.06
1000	0.94 ± 0.02	1.01 ± 0.02	1.17 ± 0.06	1.00 ± 0.02	1.15 ± 0.05
Mean	0.89 ± 0.04	0.99 ± 0.04	2.34 ± 0.32	1.01 ± 0.03	2.34 ± 0.27

Table A.7: Chapter 6: Correlation Between Money Offered and Startup Delay

d	Data FCFS	UM Data	UM NonData	UM Data+Money	UM NonData+Money
1	0.00 ± 0.02	0.00 ± 0.01	0.00 ± 0.01	-0.15 ± 0.11	-0.14 ± 0.10
2	0.00 ± 0.02	0.00 ± 0.01	0.00 ± 0.01	-0.16 ± 0.12	-0.16 ± 0.12
3	-0.01 ± 0.02	0.01 ± 0.01	0.01 ± 0.02	-0.16 ± 0.10	-0.17 ± 0.11
4	-0.01 ± 0.02	0.01 ± 0.01	0.01 ± 0.02	-0.20 ± 0.10	-0.20 ± 0.11
5	-0.01 ± 0.02	0.01 ± 0.01	0.01 ± 0.02	-0.21 ± 0.10	-0.21 ± 0.11
6	-0.01 ± 0.02	0.01 ± 0.02	0.01 ± 0.02	-0.22 ± 0.11	-0.22 ± 0.11
7	-0.01 ± 0.03	0.01 ± 0.02	0.01 ± 0.02	-0.23 ± 0.10	-0.23 ± 0.11
8	-0.01 ± 0.03	0.01 ± 0.02	0.01 ± 0.02	-0.23 ± 0.11	-0.23 ± 0.11
9	-0.01 ± 0.03	0.01 ± 0.02	0.01 ± 0.02	-0.24 ± 0.11	-0.24 ± 0.11
10	-0.02 ± 0.03	0.01 ± 0.02	0.01 ± 0.02	-0.25 ± 0.11	-0.25 ± 0.11
20	-0.02 ± 0.03	0.00 ± 0.02	0.00 ± 0.02	-0.21 ± 0.09	-0.24 ± 0.10
30	-0.01 ± 0.03	0.01 ± 0.02	0.01 ± 0.03	-0.18 ± 0.08	-0.24 ± 0.07
40	-0.01 ± 0.03	0.02 ± 0.02	0.02 ± 0.03	-0.21 ± 0.06	-0.27 ± 0.06
50	-0.01 ± 0.03	0.01 ± 0.02	0.01 ± 0.03	-0.23 ± 0.06	-0.30 ± 0.06
60	-0.01 ± 0.03	0.01 ± 0.02	0.01 ± 0.02	-0.25 ± 0.06	-0.32 ± 0.06
70	-0.01 ± 0.03	0.01 ± 0.02	0.01 ± 0.02	-0.25 ± 0.05	-0.33 ± 0.06
80	-0.01 ± 0.03	0.01 ± 0.02	0.00 ± 0.02	-0.25 ± 0.05	-0.36 ± 0.06
90	-0.01 ± 0.03	0.00 ± 0.02	0.00 ± 0.02	-0.25 ± 0.05	-0.36 ± 0.05
100	-0.01 ± 0.03	0.00 ± 0.02	0.00 ± 0.02	-0.26 ± 0.04	-0.37 ± 0.04
125	-0.01 ± 0.03	0.00 ± 0.02	0.00 ± 0.02	-0.26 ± 0.03	-0.39 ± 0.03
150	-0.01 ± 0.03	-0.01 ± 0.01	0.00 ± 0.02	-0.27 ± 0.02	-0.39 ± 0.03
175	0.00 ± 0.03	-0.02 ± 0.01	-0.01 ± 0.02	-0.29 ± 0.02	-0.39 ± 0.03
200	0.00 ± 0.03	-0.02 ± 0.01	-0.01 ± 0.01	-0.28 ± 0.02	-0.37 ± 0.03
225	0.00 ± 0.03	-0.02 ± 0.01	-0.01 ± 0.02	-0.28 ± 0.02	-0.36 ± 0.03
250	0.00 ± 0.03	-0.02 ± 0.01	-0.01 ± 0.01	-0.28 ± 0.02	-0.36 ± 0.02
275	0.00 ± 0.03	-0.02 ± 0.01	-0.02 ± 0.01	-0.29 ± 0.02	-0.35 ± 0.02
300	0.00 ± 0.03	-0.02 ± 0.01	-0.02 ± 0.01	-0.28 ± 0.02	-0.35 ± 0.02
325	0.00 ± 0.03	-0.02 ± 0.01	-0.01 ± 0.01	-0.28 ± 0.02	-0.34 ± 0.02
350	0.00 ± 0.03	-0.03 ± 0.01	-0.02 ± 0.01	-0.27 ± 0.02	-0.34 ± 0.02
375	0.00 ± 0.03	-0.03 ± 0.01	-0.02 ± 0.01	-0.27 ± 0.02	-0.34 ± 0.02
400	0.00 ± 0.03	-0.03 ± 0.01	-0.02 ± 0.01	-0.26 ± 0.02	-0.33 ± 0.02
425	0.00 ± 0.03	-0.03 ± 0.01	-0.02 ± 0.01	-0.26 ± 0.03	-0.33 ± 0.03
450	0.00 ± 0.03	-0.03 ± 0.01	-0.02 ± 0.01	-0.26 ± 0.03	-0.33 ± 0.03
475	0.00 ± 0.03	-0.03 ± 0.01	-0.02 ± 0.01	-0.26 ± 0.02	-0.33 ± 0.03
500	0.00 ± 0.03	-0.03 ± 0.01	-0.02 ± 0.01	-0.25 ± 0.02	-0.33 ± 0.03
550	0.00 ± 0.03	-0.04 ± 0.01	-0.02 ± 0.01	-0.25 ± 0.02	-0.33 ± 0.02
600	0.00 ± 0.03	-0.04 ± 0.01	-0.02 ± 0.01	-0.24 ± 0.02	-0.32 ± 0.02
650	0.00 ± 0.03	-0.04 ± 0.01	-0.03 ± 0.01	-0.25 ± 0.02	-0.31 ± 0.02
700	0.00 ± 0.03	-0.04 ± 0.01	-0.03 ± 0.01	-0.25 ± 0.02	-0.30 ± 0.02
750	0.00 ± 0.03	-0.04 ± 0.01	-0.02 ± 0.01	-0.24 ± 0.02	-0.29 ± 0.02
800	0.00 ± 0.03	-0.04 ± 0.01	-0.03 ± 0.01	-0.25 ± 0.02	-0.30 ± 0.02
850	0.00 ± 0.03	-0.04 ± 0.01	-0.03 ± 0.01	-0.25 ± 0.02	-0.30 ± 0.02
900	0.00 ± 0.03	-0.03 ± 0.01	-0.03 ± 0.01	-0.26 ± 0.02	-0.30 ± 0.02
950	0.00 ± 0.03	-0.03 ± 0.01	-0.03 ± 0.01	-0.26 ± 0.02	-0.31 ± 0.02
1000	0.00 ± 0.03	-0.03 ± 0.01	-0.03 ± 0.01	-0.27 ± 0.02	-0.32 ± 0.02
Mean	-0.01 ± 0.03	-0.01 ± 0.01	-0.01 ± 0.02	-0.24 ± 0.05	-0.30 ± 0.05