LINEAR PROGRAMMING BASED RESOURCE MANAGEMENT

LINEAR PROGRAMMING BASED RESOURCE MANAGEMENT FOR HETEROGENEOUS COMPUTING SYSTEMS

By
ISSAM AL-AZZONI, B.Eng., M.A.Sc.

A Thesis
Submitted to the School of Graduate Studies
in Partial Fulfillment of the Requirements
for the Degree
Doctor of Philosophy

McMaster University

DOCTOR OF PHILOSOPHY (2009) McMaster University
(Software Engineering) Hamilton, Ontario

TITLE: Linear Programming based Resource Management for Heterogeneous Computing Systems
AUTHOR: Issam Al-Azzoni, B.Eng., M.A.Sc. (McMaster University)
SUPERVISOR: Dr. Douglas G. Down
NUMBER OF PAGES: cviii, 108

Abstract

An emerging trend in computing is to use distributed heterogeneous computing (HC) systems to execute a set of tasks. Cluster computer systems, grids, and Desktop Grids are three popular kinds of HC systems. An important component of an HC system is its resource management system (RMS). The main responsibility of an RMS is assigning resources to tasks in order to satisfy certain performance requirements.

For cluster computer systems, we propose a new mapping heuristic which requires less state information than current heuristics. For Desktop Grids, we propose a new scheduling policy that exploits knowledge of the effective computing power delivered by the machines and the distribution of their fault times in order to improve performance. Finally, for grids, we propose a new decentralized load balancing policy which dramatically cuts down the communication overhead incurred in state information update.

The proposed resource management policies utilize the solution to a linear programming problem (LP) which maximizes the system capacity. Our simulation experiments show that these policies perform very competitively, especially in highly heterogeneous systems.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Widespread availability of low-cost, high performance computing hardware, the rapid expansion of the Internet and advances in computing networking technology have led to an increasing use of heterogeneous computing (HC) systems. An HC system is constructed by networking various machines with different capabilities and coordinating their use to execute a set of tasks. An important component of an HC system is its resource management system (RMS) which is responsible for assigning resources to tasks in order to satisfy certain performance requirements.

The first kind of HC systems we consider is the cluster computer system. Clusters of commodity computers are rapidly gaining acceptance as the preferred way to construct large computing platforms for applications with extensive computer needs (see Sterling *et al.* [66]). Such systems form the building blocks for grids which are becoming very successful in managing and organizing an institution's computing resources (see Foster *et al.* [28]).

In a cluster computer system, the RMS consists of a dedicated mapper for assigning incoming tasks to machines. The mapper immediately maps an arriving task to the system. For such HC systems, it is necessary for any mapping heuristic to stabilize the system if the system can be stabilized. Furthermore, the mapping heuristic should attempt to minimize the mean task waiting time. In addition to stability and performance concerns, the mapping heuristic should minimize the amount of state information required in mapping. As will be discussed later, this is necessary to avoid performance degradation that results from using full state information in large systems. Motivated by these requirements, we propose several mapping heuristics that perform very competitively and verify their performance using simulation.

Desktop Grids are the second kind of HC systems considered here. Desktop Grids are HC systems characterized by the non-dedication of their machines. Desktop Grids aim to harvest a large number of desktop PCs owned by individuals and whose idle cycles can be exploited to run Grid applications. Desktop Grids have recently received a lot of attention because of the success of several popular applications such as SETI@home [61].

There are several differences between Desktop Grids and cluster computer systems. One of the key differences is the dedication of machines. In cluster computer systems, the machines are assumed to be fully dedicated for executing the submitted tasks. On the other hand, in Desktop Grids, machines can also execute local tasks submitted by their owners and thus these machines are not fully dedicated for executing the Grid applications. Due to these differences, resource management for Desktop Grids is different from that of cluster computer systems. In Desktop Grids, the RMS consists of a dedicated scheduler which mainly uses pull-based scheduling (*i.e.*, a machine sends a request to the scheduler in order to get a task). Other differences will be discussed in detail later when defining the workload models for both Desktop Grids and cluster computer systems.

For Desktop Grids, we propose a new scheduling policy that exploits knowledge of the effective computing power delivered by the machines and the distribution of their fault times in order to improve scheduling performance. In the literature, several papers describe techniques that are used in predicting CPU availability and modeling machine availability. In our work, we show how such predictions can be used by the scheduler to make effective scheduling decisions.

Grids are the third kind of HC systems considered in our work. Grids can consist of a much larger number of machines than cluster computer systems. Also, grids are typically more heterogeneous and geographically dispersed. As opposed to Desktop Grids, we consider grids that consist of a dedicated infrastructure. Thus, resource management for grids should address several issues that are distinct from those of cluster computer systems and Desktop Grids.

We propose a new decentralized load balancing policy for grid resource management. In centralized policies, a central machine is dedicated as a load balancer and all tasks are submitted to the central machine. Thus, the load balancer can become a bottleneck and a single point of failure. To avoid this, decentralized load balancing policies involve all machines in load balancing and avoid the use of a central server. However, current decentralized load balancing policies suffer from significant communication overhead. This is because these policies require updating each machine with full state information. Our proposed policy performs very competitively while requiring dramatically less information and thus significantly reducing the communication overhead.

Section 1.2 defines our workload model for HC systems. Section 1.3 contains a detailed introduction to mapping heuristics for cluster computer systems. Section 1.3.1 defines the workload model for cluster computer systems, and Section 1.3.2 reviews the literature related to mapping heuristics. Section 1.4 contains a detailed introduction to scheduling policies for Desktop Grids. Section 1.4.1 defines our workload model for Desktop Grids, and Section 1.4.2 reviews the literature related to scheduling policies for Desktop Grids. Section 1.5 contains a detailed introduction to decentralized load balancing for grids. For decentralized load balancing, Section 1.5.1 gives our workload model in detail, and Section 1.5.2 discusses related literature.

1.2 A Workload Model for HC Systems

An HC system connects and coordinates various machines with different capabilities to execute a set of tasks. Let the number of machines in the system be M.

The tasks are assumed to be independent and atomic. In the literature, parallel applications whose tasks are independent are sometimes referred to as Bag-of-Tasks applications (BoT) (as in Anglano *et al.* [9]) or parameter-sweep applications (as in Casanova *et al.* [16]). Such applications are used in a variety of domains, including simulations, fractal calculations, computational biology, and computer imaging, and are becoming predominant for HC systems (see Iosup *et al.* [38] and Li and Buyya [48]).

While determining the exact task execution time on a target machine remains a challenge, there exist several techniques that can be used to estimate an expected value for the task execution time (see Rao and Huh [57]). Our policies exploit estimates on mean task execution times rather than exact execution times. Furthermore, in HC systems, tasks that belong to the same application are typically similar in their resource requirements. For example, some applications are CPU bound while others are more I/O bound. In fact, several authors have observed the high dependence of a task execution time on the application it belongs to and the machine it is running on. They argue for using application profile information to guide resource management (see Kontothanassis and Goddeau [46]). We follow the same steps and assume that the tasks are classified into groups (or classes) with identical distributions for the execution times.

It is assumed that the tasks are classified into N classes. Tasks that belong to class i arrive according to a renewal process with rate α_i. Furthermore, the execution time of a task on a machine depends on the class of the task and the machine. Let $\mu_{i,j}$ be the execution rate for tasks of class i at machine j, hence $1/\mu_{i,j}$ is the mean execution time for class i tasks at machine j. We allow $\mu_{i,j} = 0$, which implies machine j is physically incapable of executing class i tasks. Each task class can be executed by at least one machine. Let α be the arrival rate vector, where the ith element of α is α_i. Also, let μ be the execution rate matrix, having (i, j) entry $\mu_{i,j}$. We assume that $\alpha_i > 0$ for all $i = 1, \ldots, N$. Also, given any class i, we assume that there exists at least one machine j such that $\mu_{i,j} > 0$. There are further conditions on the arrival and execution time processes that are needed for our analytic results to hold (see Appendix A). Several techniques for classifying tasks and obtaining the arrival and execution rates in HC systems exist (see [46]).

1.3 Mapping Heuristics for Cluster Computer Systems

When a new task arrives to the system, the mapper uses a mapping heuristic to map the arriving task to a machine. In our work, we consider dynamic mapping heuristics (Kim *et al.* [40]). Dynamic mapping heuristics use information on the state of the system to make their mapping decisions. On the other hand, static mapping heuristics make mapping decisions independent of the state of the system.

It is necessary for any mapping heuristic to stabilize the system if the system can be stabilized. Furthermore, the mapping heuristic should attempt to minimize the mean task waiting time. In addition to addressing stability and performance, the mapping heuristic should minimize the amount of state information required in mapping. In large systems, mapping heuristics that require full state information suffer from several limitations. First, there is a significant communication overhead since the mapper needs to communicate with a large number of machines. Also, the synchronization overhead that results from requiring full state information may degrade performance. Another important problem is that the supplied information can be out of date resulting in performance degradation. As observed by Mitzenmacher [54], this is a major limitation of heuristics which attempt to exploit global information to balance load too aggressively.

Motivated by these requirements, we propose several mapping heuristics that perform very competitively and verify their performance using simulation. In particular, the Linear Programming Based Affinity Scheduling (LPAS) heuristic achieves competitive performance and at the same time requires less state information than current heuristics. Furthermore, by solving an allocation LP, the LPAS heuristic provides an explicit method to compute the maximum capacity and to compute the allocation of machines to classes.

We also describe other LPAS-related heuristics which attempt to reduce further the state information required in making mapping decisions. We introduce the Guided-LPAS heuristic, a modification of the LPAS heuristic which guarantees stability of a stabilizable system. Although the LPAS heuristic does not suffer from the root cause for instability of other heuristics, we are unfortunately not able to prove its stability.

1.3.1 Workload Model

In a cluster computer system, there is a dedicated mapper for assigning incoming tasks to machines. The mapping heuristics considered here are immediate mode heuristics [40]. In such heuristics, a mapping decision is made by the mapper as soon as a task arrives. Each new task arriving in the system is immediately assigned to one of the machines. The task can only be executed by the machine to which it is assigned. It is assumed that there is no queueing at the mapper and tasks are queued at the machines to which they are assigned. With respect to local scheduling, we assume that each machine can use any policy as long as it is non-idling.

The dynamic mapping heuristics considered here assume that the execution rates are known. A task's execution time is not known until its completion, though the task class and thus its execution rate is known to the mapper and the machines. Furthermore, in most of these heuristics, the mapper uses information supplied by the machines in making mapping decisions. Such information includes, for instance, the task queue length for each class. Thus, when a task arrives to the system, the mapper contacts the machines whose information is needed, and subsequently, the machine supplies the mapper with the requested information.

1.3.2 Literature Review

The problem of mapping tasks onto machines in HC systems is an extremely active field (for example, see Braun *et al.* [14] and Kim *et al.* [40]).

Several mapping heuristics are described and compared in Maheswaran *et al.* [51]. The model assumptions in [51] and our assumptions for the HC system are identical. However, the authors in [51] do not group tasks into classes and they assume that the expected execution time of every arriving task is known on each machine. This can be unrealistic in typical HC systems. On the other hand, we assume that the tasks are grouped into classes and only the arrival rates of each class's tasks and the execution rates of each machine for each class are known. This assumption is made in several models of cluster and grid environments (such as Franke *et al.* [29] and Kontothanassis and Goddeau [46]).

In [46], the performance of several mapping heuristics is examined on a real-world workload. One of these heuristics is similar to the MCT (Minimum Completion Time) heuristic [51]. Another heuristic is a variation on the MCT heuristic that attempts to minimize completion time while taking affinity effects into account. Experimental results show that varying the MCT heuristic to take affinity effects into account exhibits improved performance over the MCT heuristic [46].

Several dynamic mapping heuristics are proposed and compared in [40] for HC systems in which tasks have priorities and multiple soft deadlines. These heuristics are batch mode heuristics, as opposed to the immediate mode heuristics considered here. Immediate mode heuristics map an arriving task as soon as it arrives, whereas batch mode heuristics consider a subset of tasks for mapping. The workload model in [40] is identical to our workload model with the addition of priorities and deadlines associated with tasks.

Ansell *et al.* [10] develop a class of heuristics for systems having the same workload model. Such heuristics are based on the application of a policy improvement step to an optimal static heuristic. However, these heuristics are computationally intensive and may not scale well. In fact, only a small system with $N = 2$ and $M = 2$ is considered in their numerical study. Another limitation is that there is no attempt to reduce the amount of state information required in mapping. Thus, the mapper needs to obtain full state information at every mapping event.

Another heuristic is suggested in Glazebrook *et al.* [33]. The heuristic is applicable to systems having an identical workload model to the model considered here. However, the mean execution time of a task depends only on the machine (*i.e.*, for a machine j, $\mu_{i,j} = \mu_j$, for all $i = 1, \ldots, N$). Furthermore, machines may not be permanently available for service. Similar to [10], such a heuristic computes an index for each machine at every state of the system and thus may not scale to large systems.

Our model for an HC system has been studied in the context of queueing analysis. The MCT heuristic is a variation on the MinDrift rule which is shown to perform well in heavy traffic scenarios (see Stolyar [67]). Wasserman *et al.* [68] introduce a processor allocation policy which corresponds to the MCT heuristic.

1.4 Fault-Aware Scheduling Policies for Desktop Grids

A scheduling policy must support systems with a very large number of machines. Besides the natural complexity of scheduling for such large systems, the complexity is further complicated by several factors. First, Desktop Grids are characterized by very high resource volatility. In such systems, machines can fail at any time without any advance notice. Since Desktop Grids are typically based on the Internet, machines are also exposed to link failures. Furthermore, Desktop Grids are volunteer computing systems where participants voluntarily join in to execute the Grid applications. Thus, the machines of a Desktop Grid system are not dedicated (*i.e.*, machines' local jobs should have higher priority than the Grid tasks). To better cope with resource volatility, a scheduling policy must be fault-aware in the sense that it needs to exploit the knowledge of the effective computing power delivered by resources and the distribution of their fault times (if such information is available).

A second factor contributing to the complexity of scheduling for Desktop Grids is related to the heterogeneous nature of such systems. These systems interconnect a multitude of heterogeneous machines (desktops with various resources: CPU, memory, disk, etc.) to perform computationally intensive applications that have diverse computational requirements. Performance could be significantly impacted if information on task and machine heterogeneity is not taken into account by the scheduling policy. To the best of our knowledge, our work is the first to consider the problem of scheduling for heterogeneous Desktop Grids involving resource volatility.

In current Desktop Grids, the default scheduling policy is First-Come-First-Served (FCFS) (see Domingues *et al.* [27] and Kondo *et al.* [44]). It does not require any information on task arrival rates, machine execution rates or availabilities. FCFS performs well in systems with limited task heterogeneity. However, as our simulations show, its performance can be very poor in systems with high task heterogeneity and degrades rapidly as the load increases. In our work, we suggest the use of an existing policy (the G$c\mu$ policy) which has been described in the queueing literature. This policy performs much better than FCFS, but requires information on the machine execution rates. Furthermore, we develop a new policy, the Linear Programming Based Affinity Scheduling policy for Desktop Grids (LPAS_DG), which utilizes the solution to a linear programming (LP) problem that maximizes system capacity. In addition to the machine execution rates, this policy assumes knowledge of the task arrival rates and that there is a mechanism by which the scheduler detects machine failures and availabilities. Our simulation experiments show significant performance advantages for the LPAS_DG policy over the G$c\mu$ policy, especially in highly heterogeneous systems.

1.4.1 Workload Model

In our model for a Desktop Grid, there is a dedicated scheduler for assigning incoming tasks to the requesting machines. Resource management systems for Desktop Grids mainly use pull-based scheduling (see Choi *et al.* [20, 21]). In pull-based scheduling, when a machine becomes available, it sends a request to the scheduler in order to be assigned a new task for execution. Using pull-based scheduling in Desktop Grids is necessary due to the property that the machines are not dedicated. One of the results of using pull-based scheduling is that tasks queue at the scheduler side. There is no queueing at the machines; in fact, in Desktop Grids, one machine executes at most one task at a time without preemption (see Choi *et al.* [21], Domingues *et al.* [26], and Kondo *et al.* [44]). Also, in pull-based scheduling, the scheduler makes a decision as soon as it receives a request from a machine [21].

In Desktop Grids, machines can fail (or become unavailable) at any time without any advance notice [9]. If a machine fails while executing a task, then that task needs to be resubmitted to the scheduler. We assume that the scheduler becomes aware of the failure of any machine within a negligible amount of time [44]. We assume that the Desktop Grid is mainly used to execute short-lived applications [44]. These applications consist of short tasks whose mean execution times are small relative to the mean machine availability times. Hence, in such systems, we do not consider fault tolerant scheduling mechanisms such as checkpointing, migration and replication, due to their overhead.

One of the basic properties of Desktop Grids is the non-dedication of machines. When a machine is available, it may also run local jobs (*i.e.*, jobs submitted by a local user). The machines' local jobs are always given higher priority. When a machine is busy with local jobs, the result is a slowing down of the execution of the Desktop

Grid tasks submitted by the scheduler to the machine. To model the non-dedication property of machines, we use an approach similar to [9]. Let $\mu'_{i,j}$ be the nominal execution rate for tasks of class i at machine j, hence $1/\mu'_{i,j}$ is the mean nominal execution time for class i tasks at machine j. When a machine becomes available, it sends its request for a new task to the scheduler. As in [9], we assume that the machine also supplies the expected proportion of time that it is going to spend in executing the Desktop Grid tasks during its coming availability period (*i.e.*, its CPU availability). These estimates can be obtained using techniques such as those suggested by Wolski *et al.* [70] and Yang *et al.* [72]. Thus, we can define the effective execution rate $\mu_{i,j}$ for the submitted tasks as follows:

$$\mu_{i,j} = \mu'_{i,j} \times a_j$$

where a_j represents the fraction of machine j's capacity that is available for executing the Desktop Grid tasks during its coming availability period. Also, let μ be the effective execution rate matrix, having (i, j) entry $\mu_{i,j}$. As in [9, 44], once a task is submitted to a machine, the task can not be resubmitted unless a failure occurs.

1.4.2 Literature Review

A taxonomy of Desktop Grids and a survey focusing on scheduling is provided in [21]. This taxonomy is defined by three major components: the application's perspective, the resource provider's perspective, and the scheduler's perspective. With respect to our workload model, we consider applications with independent, fixed tasks that are computation-intensive. There are no deadlines associated with tasks and the tasks arrive non-deterministically to the scheduler. In terms of the resource provider's perspective, we assume that the resource providers (*i.e.*, the machines) are not dedicated to public execution and they are faulty. In terms of the scheduler's perspective, a centralized organization is assumed. The scheduler uses pull-based scheduling in which scheduling events are initiated by the resource providers.

Several fault-aware Desktop Grid scheduling policies are presented in [9] for Bag-of-Tasks applications. The proposed policies exploit fault handling mechanisms including replication and checkpointing. Furthermore, these policies exploit knowledge of the effective computing power delivered by resources and the distribution of their fault times to improve scheduling performance. The performance of the different policies is analyzed using an extensive simulation study. The policies proposed in [9] assume that the set of tasks is initially available to the scheduler, however, we assume a continuous arrival stream of tasks and that the scheduler only knows the arrival rates and execution rates (it does not need to know the entire distribution). Our work goes beyond this by addressing workloads where multiple Bag-of-Tasks applications are simultaneously submitted.

Other policies are proposed in [44]. These policies attempt to minimize the overall execution time, or the makespan, of a single parallel application. The application is assumed to consist of a number of independent tasks that is relatively small compared to the number of available resources. The policies are based on three resource selection techniques, namely resource prioritization, resource exclusion, and task replication. Even though the policies developed in [44] are designed to schedule a single application, the authors acknowledge that these policies provide key elements for designing effective "job scheduling" strategies. Furthermore, the authors planned to design scheduling policies for the scenario where multiple applications are submitted over time. In this context, our work represents a step in addressing such environments.

Several scheduling policies are suggested in [26] for institutional Desktop Grids. Institutional Desktop Grids are grids comprised of the desktop machines of an institution (academic or corporate) and thus are typically characterized by a more homogenous computing infrastructure. Similar to [9, 44], the scheduling policies are designed to minimize the turnaround time of a single Bag-of-Tasks application. The turnaround time for a Bag-of-Tasks application is defined as the elapsed time between the submission of the first task until the last task is completed.

Several papers study machine availability in Desktop Grids. In Nurmi *et al.* [56], availability data is collected from different Desktop Grid environments. Their results indicate that either a hyperexponential or Weibull distribution effectively represents machine availability in enterprise and Internet computing environments. In Kondo *et al.* [45], statistics from four real enterprise Desktop Grids are gathered in order to develop predictive models for machine availability. They distinguish between machine availability and CPU availability. The former is a binary value that indicates if the machine is reachable. Examples of machine unavailability include power failure or machine reboot. The latter is a percentage value that quantifies the fraction of the CPU that can be exploited by Desktop Grid applications.

An approach for predicting machine availability in Desktop Grids is presented in Ren *et al.* [58]. The authors apply semi-Markov process models for the prediction. They suggest a method for applying availability prediction to job scheduling. Using simulation, they show the effectiveness of their scheduling policies in large compute-bound guest applications. Our work proposes policies for short-lived applications.

A significant amount of work has been done on the measurement and characterization of CPU availability. The work in [72] includes techniques based on time series analysis for predicting CPU load at some future time point, average CPU load for some future time interval, and variation of CPU load over some future time interval. The work in [70] examines the problem of making short and medium term forecasts of CPU availability on time-shared Unix systems. Their results demonstrate the possibility of making short and medium term predictions of CPU availability despite the presence of long-range autocorrelation and potential self-similarity.

Kondo *et al.* [43] measure and characterize CPU availability in a large-scale Internet Desktop Grid. Their characterization focuses on identifying patterns of correlated availability using clustering techniques. In Rood and Lewis [59], the authors identify five availability states which capture why and how resources become unavailable over time. Their five-state availability model is motivated by the workload model of

Condor [22]. The authors characterize a Condor pool trace to develop multi-state predictors and use these predictors in developing scheduling policies for Condor (see Rood and Lewis [60]). These policies assume push-based scheduling, as opposed to the pull-based scheduling policies considered here.

1.5 Decentralized Load Balancing Policies for Grids

Even though decentralized load balancing has advantages (with respect to centralized policies) in terms of scalability and fault tolerance, the communication overhead incurred by frequent information exchange between machines represents a challenge. Current policies require updating each machine with full state information. This is problematic due to two main factors. First, the communication overhead necessary for full state information update may be prohibitive. In effect, this may cause scalability issues for grids using a decentralized load balancing approach. Second, requiring full state information may degrade performance due to the effect of outdated data.

Motivated by these issues, we propose a novel decentralized load balancing policy that performs very competitively and at the same time requires dramatically less state information. By solving an allocation LP, the Linear Programming based Affinity Scheduling load balancing policy for decentralized grids (LPAS_dec) provides an explicit method to compute the allocations of machines to tasks. Our simulations show significant performance advantages over competing policies, especially in highly heterogeneous systems.

1.5.1 Workload Model

In decentralized load balancing, a task can be submitted to any machine in the grid. Each machine is responsible for the assignment of its locally submitted tasks to one of the grid machines. Upon the arrival of a task to a machine, the machine immediately makes a decision on whether to execute the task locally or send it for execution on another machine. After that, the task can only be executed by the machine to which it is assigned. Thus, a task can not be migrated more than once. Similarly, several researchers assume a migration limit of one as task migration is often difficult in practice and there are no significant benefits of using higher migration limits (see Lu *et al.* [50] and Shah *et al.* [62]).

We consider dynamic load balancing policies. These policies, as opposed to static policies, attempt to exploit dynamic state information to optimize performance. In order to do that, certain types of information need to be exchanged among the machines, *e.g.* task queue lengths, machine execution rates, and so forth. In grids, there is no efficient state-broadcast mechanism [50]. Other approaches for information exchange, such as state-polling, are also problematic in practice (see Gu *et al.* [34], Lu *et al.* [50], and Werstein *et al.* [69]). To minimize the overhead of information collection, we assume that state information exchange is done by mutual information feedback [50]. Thus, when a machine j_1 needs the local state information of another machine j_2, then it sends a request message to j_2 which in turn sends back a reply message. Both the request and reply messages may embed other local state information as dictated by the load balancing policy.

1.5.2 Literature Review

Several authors have suggested decentralized load balancing policies for grids. In general, a decentralized load balancing policy should address the following:

1. Information exchange policy. The information exchange policy is concerned with how to update each machine with the state information of other machines. Two techniques for information exchange were discussed earlier: state polling and mutual information feedback. Several load balancing policies that use state polling are presented in [34, 62, 69]. The LPAS_dec policy uses mutual information feedback, as do the policies presented in Arora *et al.* [13], Lu *et al.* [50], and Rao and Huh [57].

2. Transfer policy. The transfer policy address the question of when to balance the load. Some policies balance the load only upon task arrivals, including the LPAS_dec policy and [13, 57, 62]. Other policies [69] are threshold-based *e.g.*, only when the load on a machine exceeds a certain threshold, load balancing is triggered. Some load balancing policies use a combination of both techniques [34, 50].

3. Selection Policy. On the event of load balancing, the selection policy determines which tasks to migrate. Policies, such as the LPAS_dec policy and [57, 62], which balance the load upon task arrivals migrate only the arriving task. Some policies, however, migrate additional tasks, such as [13]. Other policies which use a threshold-based transfer policy rank the queued tasks based on certain criteria and only migrate the highest ranking tasks (see [34, 50]).

4. Placement policy. On the event of load balancing and task migration, the placement policy determines the machines into which these tasks are to be migrated. Some policies use the expected completion time as a metric for the placement policy (including the LPAS_dec policy, the LBA policy [62], and the IDP policy [50]). Other policies use the expected load on the target machines and only migrate tasks to the least loaded machines (including [34]).

Performance monitoring tools such as NWS [71] and MonALISA [47] can be used to provide dynamic information on the state of the grid system. Furthermore, these tools anticipate the future performance behaviour of an application including task arrival and machine execution rates.

1.6 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 discusses mapping heuristics for cluster computing systems. Chapter 3 discusses scheduling policies for

Desktop Grids. Chapter 4 discusses decentralized load balancing for grids. Each chapter first discusses several relevant policies, then presents our linear programming based policy, and finally discusses simulation results. Chapter 5 concludes the thesis and outlines suggestions for possible future work. Appendix A contains detailed proofs for the theorems in Chapter 2.

Chapter 2

Mapping Heuristics for Cluster Computer Systems

In this chapter, we describe the LPAS-related mapping heuristics for cluster computer systems. Section 2.1 describes several related mapping heuristics. Section 2.2 introduces the LPAS heuristic and includes simulation results that compare the performance of various mapping heuristics. Other LPAS-related heuristics that attempt to reduce further the required state information for mapping are discussed in Section 2.3. The Guided-LPAS heuristic is introduced in Section 2.4. Appendix A contains detailed proofs for several results discussed in this chapter. Contents of this chapter appear in Al-Azzoni and Down [3, 5] and He *et al.* [36].

2.1 Mapping Heuristics

A mapper using the MET (minimum execution time) heuristic assigns an incoming task to the machine that has the least expected execution time for the task [51]. Thus, when a new task of class i arrives, the mapper assigns it to a machine j such that $j \in \arg\min_{j'} 1/\mu_{i,j'}$. Ties are broken arbitrarily; for instance, a mapper could pick the machine with the smallest index j when more than one machine has the least expected execution time. The MET heuristic does not require the machines to send their expected completion times back to the mapper as tasks arrive, thus the MET heuristic has the advantage of requiring limited communication between the mapper and machines. However, this heuristic can cause severe load imbalance to a degree that the system is unstable. For example, consider a system with one arrival stream with rate $\alpha_1 = 6$, and two machines with execution rates $\mu_{1,1} = 5$ and $\mu_{1,2} = 3$, respectively. This system will suffer from load imbalance causing instability if the MET heuristic is used, as no tasks are sent to machine 2. It is easy to see that the system can be stabilized with the given value of α_1.

The MCT (minimum completion time) heuristic assigns an arriving task to the machine that has the earliest expected completion time [51]. Several existing resource management systems, *e.g.* NetSolve [12] and SmartNet [30, 31], use the MCT heuristic or other heuristics that are based on the MCT heuristic. The mapper examines all machines in the system to determine the machine with the earliest expected completion time.

There are several limitations for the MCT heuristic. First, the mapper requires full state information since it needs to obtain the queue lengths of all machines in the system. Second, the MCT heuristic suffers from its lack of any foresight about task heterogeneity. It might assign an arriving task to a poor machine which minimizes the task's completion time, yet causes problems for future arrivals. Consider the following system with $M = 7$ and $N = 4$. We will refer to this system as System $2.H$. The arrival and execution rates are given by $\alpha = [8.5\ 8.5\ 9.6\ 8.5]$ and

$$\mu = \begin{bmatrix} 5 & 5.02 & 4.95 & 0.001 & 4.7 & 5.2 & 5.25 \\ 0.001 & 5.09 & 4.9 & 4.92 & 5 & 5.13 & 5.14 \\ 4.45 & 5 & 0.001 & 4.45 & 4.9 & 5 & 5.1 \\ 5.02 & 4.95 & 5 & 5.02 & 5.25 & 4.75 & 0.001 \end{bmatrix}.$$

The system contains a few machines that have very poor performance when executing tasks that belong to particular classes. While such values would most likely not arise in practice, we have chosen these values to emphasize the point that assigning a task to a machine that is very poor executing its class may result in significant performance degradation. Since the MCT heuristic maps each arriving task to the machine that minimizes its expected completion time, it may assign an arriving task of class i to a machine j that is very poor executing class i tasks. Since the MCT heuristic does not prevent this from happening, it can result in very poor performance. Other heuristics, including the LPAS heuristic, perform much better than the MCT heuristic in such cases since they avoid mapping an arriving task to its minimum expected completion time machine that could do better for future task arrivals. Simulation results for System $2.H$ are shown later.

Furthermore, the tendency of the MCT heuristic to make mapping decisions based on the immediate marginal improvement in completion time for an arriving task may be problematic. In fact, using the MCT heuristic may result in an unstable system even though the system can be stabilized. The instability of the MCT heuristic is demonstrated in Sharifnia [63] by considering the following system with $M = 2$ and $N = 4$. The arrival and execution rates are given by $\alpha = [10\ 10\ 25\ 40]$ and

$$\mu = \begin{bmatrix} 15.38 & 0 \\ 16.67 & 16.67 \\ 0 & 50 \\ 16.67 & 200 \end{bmatrix},$$

respectively. Simulation experiments show the instability of the MCT heuristic for such a system. Other heuristics, including the LPAS heuristic, do not suffer from this limitation. In fact, the LPAS heuristic does stabilize this system.

In order to address the limitations of the MCT heuristic, the k-percent best (KPB) heuristic [51] identifies for each class a subset consisting of the $(kM/100)$ best machines based on the execution times for the class, where $100/M \leq k \leq 100$. Let S_i^k be the set of the $\lfloor kM/100 \rfloor$ machines that have the smallest expected execution time for class i tasks. The mapper assigns an arriving class i task to the machine in the subset S_i^k that has the earliest expected completion time. Define $\overline{k} = \lfloor kM/100 \rfloor$ to be the number of machines considered by the KPB heuristic.

The KPB heuristic does not attempt only to assign an arriving task to a superior machine based on execution times, it also attempts to avoid assigning an arriving task to a machine that could do better for tasks that arrive later. As discussed earlier, this foresight is not present in the MCT heuristic. Another advantage of the KPB heuristic is that the mapper needs only to communicate with a subset of the machines for each class, rather than with all machines in the system. Thus, the mapper requires less state information than the MCT heuristic.

While the KPB heuristic succeeds in reducing the required state information for mapping, setting its parameter (k) may be problematic and is done in an ad-hoc manner in [51]. Instability or severe performance degradation can result if inappropriate values for k are used. Also, the KPB heuristic maps each class to the same number of machines, which may not be desirable.

As will be discussed in the next section, the LPAS heuristic builds on the idea of the KPB heuristic. Instead of mapping each class to a fixed number of machines, the LPAS heuristic maps each class to a different set of machines based on the solution of an allocation LP. Furthermore, by solving an allocation LP, the LPAS heuristic provides an explicit method to compute the maximum capacity and to compute the allocation of machines to classes. This has the advantage of requiring dramatically less state information while at the same time achieving competitive performance levels. The LPAS heuristic maps each class to a much smaller number of machines than the MCT heuristic. Furthermore, the LPAS heuristic provides a systematic way to choose the machines.

2.2 The LPAS Heuristic

2.2.1 Overview

Our proposed heuristic is similar to the KPB heuristic in that the mapper needs only to consider a subset of the machines for each class, however, the determination of this subset requires solving a linear programming problem (LP) (Andradóttir *et al.* [7]). Then, the mapper assigns the task to the machine that has the earliest expected completion time in the subset.

The LPAS heuristic requires solving the following allocation LP, where the decision variables are λ and $\delta_{i,j}$ for $i = 1, \ldots, N$, $j = 1, \ldots, M$ (recall that $\mu_{i,j}$ and α_i are the execution rates and arrival rates for the system, respectively). The variables $\delta_{i,j}$ are to be interpreted as the proportional allocation of machine j to class i.

$$\max \lambda$$

$$\text{s.t.} \sum_{j=1}^{M} \delta_{i,j}\mu_{i,j} \geq \lambda\alpha_i, \qquad \text{for all } i = 1, \ldots, N, \tag{2.1}$$

$$\sum_{i=1}^{N} \delta_{i,j} \leq 1, \qquad \text{for all } j = 1, \ldots, M, \tag{2.2}$$

$$\delta_{i,j} \geq 0, \qquad \text{for all } i = 1, \ldots, N, \text{ and } j = 1, \ldots, M. \tag{2.3}$$

The left-hand side of (2.1) represents the total execution capacity assigned to class i by all machines in the system. The right-hand side represents the arrival rate of tasks that belong to class i scaled by a factor of λ. Thus, (2.1) enforces that the total capacity allocated for a class should be at least as large as the scaled arrival rate for that class. This constraint is needed to have a stable system. The constraint (2.2) prevents overallocating a machine and (2.3) states that negative allocations are not allowed.

Let λ^* and $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, be an optimal solution to the allocation LP. The allocation LP always has a solution, since no lower bound constraint is put on λ. However, the physical meaning of λ^* requires that its value be at least one. In this case, $1/\lambda^*$ is interpreted as the long-run utilization of the busiest machine.

The value λ^* can also be interpreted as the maximum capacity of the system. We define the maximum capacity as follows. Consider a system with given values for α_i ($i = 1, \ldots, N$) and λ^*. If $\lambda^* \leq 1$, then the system is unstable. Thus, the system will be overloaded and tasks queue indefinitely. If, however, $\lambda^* > 1$, then the system can be stabilized even if each arrival rate is increased by a factor less than or equal to λ^* (*i.e.*, the same system with arrival rates $\alpha'_i \leq \lambda^* \alpha_i$, for all $i = 1, \ldots, N$, can be stabilized). In this case, the values $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, can be interpreted as the long-run fraction of time that machine j should spend on class i in order to stabilize the system under maximum capacity conditions. Let δ^* be the machine allocation matrix where the (i, j) entry is $\delta^*_{i,j}$.

The following theorems summarize these stability results (the proofs are provided in Appendix A). For $j = 1, \ldots, M$, we let $W_j(t)$ be the total workload at machine j at time t which is defined as the cumulative amount of time that it takes machine j to execute all tasks present in its queue at time t. Let $W(t)$ be a vector with jth element $W_j(t)$.

Theorem 2.2.1 *If $\lambda^* > 1$, then the system can be stabilized. More specifically, the workload process ($\{W(t)\}$) converges to a steady-state distribution as $t \to \infty$.*

Theorem 2.2.2 *If $\lambda^* < 1$, then the system can not be stabilized. Thus, as $t \to \infty$, tasks queue indefinitely regardless of the implemented mapping heuristic.*

The LPAS heuristic can be stated as follows. Given a system, solve the allocation LP to find $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. When a new task of class i arrives, let

S_i denote the set of machines whose $\delta^*_{i,j}$ is not zero ($S_i = \{j : \delta^*_{i,j} \neq 0\}$). The mapper assigns the task to the machine $j \in S_i$ that has the earliest expected completion time among the subset of machines S_i. Again, ties are broken arbitrarily. Note that the LPAS heuristic does not use the actual values for $\{\delta^*_{i,j}\}$, beyond differentiating between the zero and nonzero elements. We must solve the allocation LP to know where the zeros are.

The LPAS heuristic considers both the arrival rates and execution rates and their relative values in deciding the allocation of machines to tasks. Furthermore, by solving the allocation LP, the LPAS heuristic provides a systematic approach for setting parameters that guarantee the stability of a stabilizable system. This is an advantage over the KPB heuristic where figuring the correct value for $\overline{k}$ may not be trivial. The KPB heuristic maps each class to $\overline{k}$ machines independent of the class, whereas the LPAS heuristic maps each class to a different subset of the machines based on the solution of the allocation LP. The following example clarifies this point and provides some intuition for the LPAS heuristic.

Consider a system with two machines and two classes of tasks ($M = 2$, $N = 2$). The arrival and execution rates are as follows:

$$\alpha = \begin{bmatrix} 2.45 & 2.45 \end{bmatrix} \text{ and } \mu = \begin{bmatrix} 9 & 5 \\ 2 & 1 \end{bmatrix}.$$

Solving the allocation LP gives $\lambda^* = 1.0204$ and

$$\delta^* = \begin{bmatrix} 0 & 0.5 \\ 1 & 0.5 \end{bmatrix}.$$

A mapper using the LPAS heuristic maps all arriving tasks that belong to class 1 to machine 2. At the times of their arrivals, tasks that belong to class 2 are mapped to the machine, either machine 1 or 2, that has the earliest expected completion time.

Even though machine 1 has the fastest rate for class 1, the mapper does not assign any class 1 tasks to it. Since the system is highly loaded, and since $\frac{\mu_{1,1}}{\mu_{2,1}} < \frac{\mu_{1,2}}{\mu_{2,2}}$ and $\alpha_1 = \alpha_2$, the performance is improved significantly if machine 1 only executes class 2 tasks. In fact, the performance of the LPAS heuristic is better than that of the MCT heuristic. For this particular system, both the MET heuristic and the KPB heuristic (with $\overline{k} = 1$) result in unstable systems. This is because both heuristics map class 2 tasks to machine 1 only, and the system will be unstable since $\alpha_2 > \mu_{2,1}$.

In the LPAS heuristic, the mapper considers a subset of the machines for each class. Ideally, the size of each subset should be much smaller than M. Similar to the KPB heuristic, this has the advantage of requiring less communication between the mapper and the machines. Furthermore, degradation in performance due to outdated information is reduced. To achieve this, the δ^* matrix should contain a large number of elements that are equal to zero. In fact, there could be many optimal solutions to an allocation LP, and an optimal solution with a larger number of zeros in the δ^* matrix is preferred. The following proposition gives the number of zero elements in the δ^* matrix that could be achieved (the proof can be found in [7]):

Proposition 2.2.1 *There exists an optimal solution to the allocation LP with at least $NM + 1 - N - M$ elements in the δ^* matrix equal to zero.*

Ideally, the number of zero elements in the δ^* matrix should be $NM + 1 - N - M$. If the number of zero elements is greater, the LPAS heuristic would be significantly restricted in shifting workload between machines resulting in performance degradation. Also, solutions that result in degenerate cases should be avoided. For example, if the δ^* matrix contains no zeros at all, then the LPAS heuristic reduces to the MCT heuristic. Throughout this chapter, we use the unique optimal solution in which the δ^* matrix contains exactly $NM + 1 - N - M$ zeros.

The LPAS heuristic can be considered as a dynamic mapping heuristic. As the heuristic only involves solving an LP, it is suited for scenarios when machines are added and/or deleted from the system. On each of these events, one needs to simply solve a new LP and continue with the new values.

2.2.2 Simulation Results

Overview

We use simulation to compare the performance of the mapping heuristics. The task arrivals are modeled by independent Poisson processes, each with rate α_i, $i = 1, \ldots, N$. Several distributions are used for execution times. Unless otherwise stated, the execution times are exponentially distributed.

Each simulation experiment models a particular system, characterized by the values of M, N, α_i, and $\mu_{i,j}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. Each experiment simulates the execution of the corresponding system for 20,000 time-units. Each experiment is repeated 30 times. All confidence intervals are at the 95%-confidence level.

There are several performance metrics that could be used to compare the performance of the mapping heuristics [51]. We have chosen the long-run average number of tasks in the system, $\bar{Q}$, as a metric for performance comparison. This includes the tasks that are waiting for execution at a particular machine and tasks that are executing.

Table 2.1 lists simulation results for different systems (these systems are discussed shortly). For each system, the table shows the 95%-confidence interval for $\bar{Q}$ when the corresponding mapping heuristic is used. If a system becomes unstable due to the mapping heuristic used by its mapper, the table just indicates that the system is unstable. Since the MET heuristic results in unstable systems in most of the systems in Table 2.1, we do not include it here. The table also shows the results of using the KPB heuristic with different values for $\overline{k}$.

In these simulation experiments, we assume that a First-Come-First-Serve (FCFS) scheduling policy is used by the machines. Thus, in this case, the expected completion time of a class i arrival at machine j is given by

$$\frac{1}{\mu_{i,j}} + \sum_{i'=1}^{N} \frac{Q_{i',j}}{\mu_{i',j}},$$

where $Q_{i',j}$ is the number of tasks of class i that are waiting or executing at machine j, at the time of the task arrival.

Small Systems

System 2.A in Table 2.1 is the system discussed in Section 2.2.1. This is a highly loaded system as shown by the large values for $\bar{Q}$. As shown in the table, the MCT heuristic performs poorly with respect to the LPAS heuristic. This is because the MCT heuristic assigns some class 1 tasks to machine 1, although it is more advantageous to dedicate machine 1 for class 2 tasks.

System 2.B is another small system, where $M = 2$ and $N = 2$. The arrival and execution rates are as follows:

$$\alpha = \begin{bmatrix} 5 & 8 \end{bmatrix} \text{ and } \mu = \begin{bmatrix} 8 & 3 \\ 4 & 10 \end{bmatrix}.$$

Solving the allocation LP gives $\lambda^* = 1.3333$ and

$$\delta^* = \begin{bmatrix} 0.8333 & 0 \\ 0.1667 & 1 \end{bmatrix}.$$

As indicated by the nonzero elements of the δ^* matrix, the LPAS heuristic assigns all class 1 tasks to machine 1. Thus, machine 2 becomes dedicated to execute class 2 tasks. This results in improved performance since, in this case, class 2 tasks have a higher arrival rate and they run much faster on machine 2 than on machine 1.

Large Systems

System 2.$C1$ is a large system with $M = 30$ and $N = 3$. The machines are grouped into four groups, and each group consists of machines with identical performance. Thus, if two machines are in the same group, then they have the same execution rates. Table 2.2 shows the execution rates of the groups.

Groups P, Q, R, and S, consist of 10 machines, 9 machines, 6 machines, and 5 machines, respectively. As Table 2.2 shows, the groups vary in performance. For

Table 2.1: Comparison of the mapping heuristics

System	MCT	KPB	LPAS
2.A	$(85.68, 110.23)$	$\underline{\overline{k} = 1}$ Unstable	$(62.56, 82.01)$
2.B	$(20.05, 21.10)$	$\underline{\overline{k} = 1}$ $(5.65, 5.73)$	$(5.21, 5.26)$
2.$C1$	$(53.99, 54.98)$	$\underline{\overline{k} = 14}$ $(75.26, 79.13)$	$(47.39, 47.72)$
2.D	$(22.68, 23.21)$	$\underline{\overline{k} = 2}$ $(14.75, 14.89)$ $\underline{\overline{k} = 3}$ $(11.00, 11.04)$	$(10.55, 10.59)$
2.E	$(27.71, 28.20)$	$\underline{\overline{k} = 5}$ $(51.65, 55.60)$	$(36.54, 37.07)$
2.$F1$	$(19.09, 19.44)$	$\underline{\overline{k} = 4}$ $(20.77, 21.07)$	$(28.71, 29.05)$
2.$F2$	$(46.36, 49.49)$	$\underline{\overline{k} = 4}$ $(73.44, 81.75)$	$(34.27, 34.89)$
2.G	$(37.91, 40.43)$	$\underline{\overline{k} = 4}$ $(42.21, 43.54)$	$(42.05, 43.09)$
2.H	$(3648.48, 4086.54)$	$\underline{\overline{k} = 5}$ $(888.62, 1319.97)$	$(131.08, 150.15)$
2.$I1$	$(64.20, 66.32)$	$\underline{\overline{k} = 14}$ $(86.65, 94.15)$	$(50.83, 38)$
2.$I2$	$(41.56, 41.82)$	$\underline{\overline{k} = 14}$ $(53.69, 55.19)$	$(40.57, 40.69)$

Table 2.2: Execution rates for System 2.$C1$

	Group			
Task	P	Q	R	S
1	8	4	4	4
2	1	4	1	2
3	4	2	8	4

instance, a machine in group P is twice as fast as a machine in group S on tasks of class 1, however, for tasks of class 2, the opposite is true. The arrival rates are given by $\alpha = [45\ 45\ 40]$.

Since System 2.$C1$ consists of some identical machines, there are an infinite number of optimal solutions to the allocation LP. To better capture machine homogeneity of the system, it is desirable to use the unique optimal solution in which machines that belong to the same group have identical values for $\delta^*_{i,j}$. To do this, we solve the allocation LP corresponding to the following system:

$$N = 3, M = 4, \alpha = [45\ 45\ 40], \text{and } \mu = \begin{bmatrix} 80 & 36 & 24 & 20 \\ 10 & 36 & 6 & 10 \\ 40 & 18 & 48 & 20 \end{bmatrix}.$$

Solving the modified allocation LP gives $\lambda^* = 1.1146$ and

$$\delta^* = \begin{bmatrix} 0.6270 & 0 & 0 & 0 \\ 0.3730 & 1 & 0.0712 & 1 \\ 0 & 0 & 0.9288 & 0 \end{bmatrix}.$$

Thus, for System 2.$C1$, we use the δ^* matrix represented in Table 2.3. In this particular solution, machines that belong to the same group have identical values for $\delta^*_{i,j}$. Note that the δ^* matrix has 44 elements that are equal to zero. However, note that based on Proposition 2.2.1, there exists another optimal solution to the allocation LP with 58 elements in the δ^* matrix that are equal to zero.

Table 2.3: The machine allocation matrix for System 2.$C1$

	Group			
Task	P	Q	R	S
1	0.6270	0	0	0
2	0.3730	1	0.0712	1
3	0	0	0.9288	0

As shown in Table 2.1, the LPAS heuristic achieves the best results. Note that the KPB heuristic is unstable for $\overline{k} < 14$.

Task and Machine Heterogeneity

Systems 2.D through 2.G model different kinds of system heterogeneity. Machine heterogeneity refers to the average variation along the rows of the execution rate matrix, and similarly task heterogeneity refers to the average variation along the columns (see Armstrong [11]). Heterogeneity can be classified into high heterogeneity and low heterogeneity. Based on this, we simulate the following four categories for heterogeneity [11, 51]: (a) high task heterogeneity and high machine heterogeneity (HiHi), (b) high task heterogeneity and low machine heterogeneity (HiLo), (c) low task heterogeneity and high machine heterogeneity (LoHi), and (d) low task heterogeneity and low machine heterogeneity (LoLo).

System 2.D models a HiHi system with $M = 7$ and $N = 4$. The arrival and execution rates are given by $\alpha = [12.5\ 12\ 12.5\ 12]$ and

$$\mu = \begin{bmatrix} 4.5 & 2 & 9.5 & 6.2 & 10.25 & 2.25 & 3.95 \\ 6.2 & 4.5 & 6 & 2 & 4.2 & 5.9 & 10.25 \\ 9.5 & 6.5 & 4 & 10 & 5.9 & 2.25 & 3.95 \\ 2.25 & 10 & 2 & 3.95 & 1.75 & 10 & 1.75 \end{bmatrix}.$$

Solving the allocation LP gives $\lambda^* = 1.3449$ and

$$\delta^* = \begin{bmatrix} 0 & 0 & 0.6907 & 0 & 1 & 0 & 0 \\ 0.2830 & 0 & 0.3093 & 0 & 0 & 0.3861 & 1 \\ 0.7170 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0.6139 & 0 \end{bmatrix}.$$

For System 2.D, the LPAS heuristic outperforms the other heuristics. It maps the tasks of each class to at most two machines, except for class 2 tasks that are mapped to four machines. The LPAS heuristic exhibits better performance than the KPB heuristic with $\overline{k} = 3$.

System 2.E is a LoHi system. The system contains 7 machines and 4 classes. The arrival and execution rates are given by $\alpha = [10\ 10\ 8\ 8]$ and

$$\mu = \begin{bmatrix} 2.2 & 7 & 10.25 & 1 & 5.7 & 0.5 & 12 \\ 1.95 & 7.05 & 9.78 & 0.95 & 5.65 & 0.56 & 11.85 \\ 2 & 7.25 & 10.02 & 0.98 & 5.75 & 0.67 & 11.8 \\ 2.05 & 6.75 & 9.99 & 1.02 & 5.82 & 0.49 & 12.05 \end{bmatrix}.$$

Solving the allocation LP gives $\lambda^* = 1.0844$ and

$$\delta^* = \begin{bmatrix} 1 & 0 & 0.8433 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.9151 \\ 0 & 1 & 0.0754 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0.0813 & 1 & 1 & 0 & 0.0849 \end{bmatrix}.$$

The MCT heuristic has the best performance for System 2.E. This is not an unexpected result as suggested by the following argument. Due to the very low task heterogeneity of system 2.E, one can think of it as a system with one class of arriving tasks ($\alpha = [36]$) and the execution rate of each machine is the average of the execution rates of the four classes in System 2.E on the machine, $\mu = [2.05\ 7.0125\ 10.01\ 0.9875\ 5.73\ 0.555\ 11.925]$. In this case, assigning an arriving task to the machine that has the minimum expected completion time (the MCT heuristic) is the best strategy. In fact, solving the allocation LP corresponding to the modified system above results in the following value for δ^*: $[1\ 1\ 1\ 1\ 1\ 1\ 1]$. Thus, in this case, the LPAS heuristic reduces to the MCT heuristic.

Even though the MCT heuristic is the best heuristic for System 2.E, the LPAS heuristic has the advantage of mapping each class to a smaller number of machines. The LPAS heuristic performs much better than the KPB heuristic even for $\overline{k} = 5$. The KPB heuristic is unstable for $\overline{k} < 5$.

Systems 2.$F1$ and 2.$F2$ are HiLo systems ($M = 7$, $N = 4$). Both have the same execution rates and only differ in the arrival rate vector α. The arrival rate vector for System 2.$F1$ is $\alpha = [4\ 8\ 10\ 10]$, and for System 2.$F2$ it is given by $\alpha = [7\ 7\ 7\ 7]$. For both systems, the execution rate matrix is given by

$$\mu = \begin{bmatrix} 2 & 2.5 & 2.25 & 2 & 2.2 & 1.75 & 2.25 \\ 4.5 & 4 & 4.2 & 4 & 3.8 & 3.9 & 3.95 \\ 6 & 6.2 & 6.25 & 6 & 5.75 & 5.9 & 6.05 \\ 10 & 10.25 & 10.5 & 9.5 & 10.25 & 10.25 & 10 \end{bmatrix}.$$

For System 2.$F1$, solving the allocation LP gives $\lambda^* = 1.1331$ and

$$\delta^* = \begin{bmatrix} 0 & 1 & 0 & 0 & 0.8946 & 0 & 0.0285 \\ 1 & 0 & 1 & 0.0911 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.9089 & 0 & 0 & 0.9715 \\ 0 & 0 & 0 & 0 & 0.1054 & 1 & 0 \end{bmatrix}.$$

For System 2.$F2$, solving the allocation LP gives $\lambda^* = 1.0798$ and

$$\delta^* = \begin{bmatrix} 0 & 1 & 0.2704 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0.7282 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.0014 & 1 & 0 & 0.2626 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7374 & 0 \end{bmatrix}.$$

Due to the very low machine heterogeneity of both systems, one can think of them as consisting of identical machines. The LPAS heuristic achieves the best performance in many such systems as in 2.$F2$.

System 2.G is a LoLo system with $M = 7$ and $N = 4$. The arrival and execution rates are given by $\alpha = [8\ 9\ 7\ 10]$ and

$$\mu = \begin{bmatrix} 5 & 5.05 & 4.95 & 4.98 & 4.7 & 5.2 & 5.25 \\ 5.25 & 5.09 & 4.9 & 4.92 & 5 & 5.13 & 5.14 \\ 4.45 & 5 & 4.9 & 4.45 & 4.9 & 5 & 5.1 \\ 5.02 & 4.95 & 5 & 5.02 & 5.25 & 4.75 & 5 \end{bmatrix}.$$

Solving the allocation LP gives $\lambda^* = 1.0557$ and

$$\delta^* = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0.6182 \\ 1 & 0.8352 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1648 & 0.9426 & 0 & 0 & 0 & 0.3818 \\ 0 & 0 & 0.0574 & 1 & 1 & 0 & 0 \end{bmatrix}.$$

For System 2.G, the MCT heuristic has slightly better performance than the other heuristics. The KPB heuristic ($\overline{k} = 4$) has performance close to that of the LPAS heuristic, however, the mapper is required to obtain the expected completion times from four machines at each task arrival as compared to at most three machines in the case of the LPAS heuristic.

Special Systems

Consider System 2.H defined in Section 2.1. As explained earlier, since the MCT heuristic does not have any foresight on task heterogeneity, it may assign an arriving task to a machine that minimizes the task's expected completion time, yet it is very poor executing the task's class. This results in significant performance degradation as shown in Table 2.1. The LPAS heuristic is the best heuristic for System 2.H. The KPB heuristic performs poorly and is only stable for $\overline{k} \geq 5$. For $\overline{k} < 5$, instability results. For $\overline{k} \geq 5$, the system becomes stable, however the performance is poor.

Other Execution Time Distributions

To test the effect of execution time distribution on the performance of the mapping heuristics, all of the previous experiments were re-run with non-exponential execution time distributions. In particular, two distributions were used to study lower and higher variances than the exponential case: the first is a constant execution time of size $\frac{1}{\mu_{i,j}}$ for machine j executing class i tasks, and the second is a hyper-exponential distribution with mean $\frac{1}{\mu_{i,j}}$ for the execution times and twice the variance as the exponential case.

Our results indicate that the relative performance of the heuristics is not affected by the execution time distribution. System $2.I1$ has the same configuration as system $2.C1$, but with a hyper-exponential execution time distribution. System $2.I2$ also has the same configuration as system $2.C1$, but with constant execution times. Table 2.1 shows the performance of the different mapping heuristics for Systems $2.I1$ and $2.I2$. For the KPB heuristic, both systems are unstable for $\overline{k} < 14$.

2.3 Other LPAS-Related Heuristics

2.3.1 Overview

In this section, we describe other LPAS-related heuristics which attempt to reduce further the state information required in making mapping decisions.

To compare mapping heuristics in terms of state information required for mapping, we use the discount metric defined in [36]. Let N_s be the average number of machines from which a heuristic acquires information for each arrival. The discount of the mean required state information (over full information) for a mapping decision is then defined by

$$(2.4)\ Discnt = \left(1 - \frac{N_s}{M}\right) \times 100\%.$$

Now, consider the heuristics introduced in Section 2.1. For the MET heuristic, the mapper need not contact any machine in making mapping decisions and thus $N_s = 0$. For the MCT heuristic, assuming that all $\mu_{i,j}$ are positive, the mapper needs to acquire state information from all of the machines and thus $N_s = M$. For the KPB heuristic, the mapper needs to acquire state information from $\overline{k}$ machines (assuming that all $\mu_{i,j}$ are positive). For the LPAS heuristic, the mean number of machines from which the mapper acquires information for each arrival is

$$(2.5)\quad N_s = \sum_{i=1}^{N} \frac{\alpha_i}{\tilde{\alpha}} |S_i|,$$

where $\tilde{\alpha} = \sum_{i=1}^{N} \alpha_i$.

The LPAS$-2/k$ Heuristic

One way to reduce further the state information required in making mapping decisions is to choose for each arrival of class i just two machines from the set S_i and then compare that pair in terms of the expected completion times. The LPAS$-2/k$ heuristic is stated as follows: A class i arrival is mapped to one of the two machines (j_1, j_2) chosen from S_i which has shorter expected completion time. If $|S_i| > 2$, machine j_1 is first chosen from S_i with probability $p_{j_1} = \frac{\delta^*_{i,j_1}\mu_{i,j_1}}{\lambda^*\alpha_i}$. Then, machine j_2 is chosen from $S_i \setminus \{j_1\}$ with probability $p_{j_2} = \frac{\delta^*_{i,j_2}\mu_{i,j_2}}{\lambda^*\alpha_i - \delta^*_{i,j_1}\mu_{i,j_1}}$.

The mean number of machines from which the LPAS$-2/k$ heuristic acquires state information for each arrival is given by

$$N_s = 2 \sum_{i:|S_i|>1} \frac{\alpha_i}{\tilde{\alpha}} + \sum_{i:|S_i|=1} \frac{\alpha_i}{\tilde{\alpha}}.$$

It is noted that, in the worst case, the LPAS$-2/k$ heuristic acquires state information from two machines for each arrival, and thus the discount of the average required state information is $(M-2)/M \times 100\%$. This implies that the discount increases as the number of machines grows, independently of the structure of δ^*. Note, however, that even though the LPAS$-2/k$ heuristic requires less state information than the LPAS heuristic, one needs to know the values for $\delta^*_{i,j}$ (rather than just whether they are zero or not).

Consider a system with $N = 1$ and M arbitrary. Assume that the execution times are exponentially distributed and $\mu_{1,j} = 1$ for all $j = 1, \ldots, M$. Also, assume that the arrival process is Poisson with rate $M\alpha_1$, where $\alpha_1 < 1$. In this case, $\delta^*_{1,j} = 1$ for all $j = 1, \ldots, M$. Thus, using the LPAS$-2/k$ heuristic, an arrival randomly (with equal probabilities) chooses two of the machines and joins the queue of the machine with the shorter queue length. Mitzenmacher [55] analyzed such a system and found that when α_1 approaches 1, there is an exponential improvement in the mean waiting time (over choosing only one machine randomly), while increasing the number of choices for an arrival results in only a constant improvement over two choices. This suggests that a similar degree of improvement might be expected for the LPAS$-2/k$ heuristic over a static mapping heuristic, although the "power of two choices" has not been analyzed rigorously for heterogeneous systems [36].

The LP–Static Heuristic

The LP–Static heuristic requires no state information in making mapping decisions. We define it here to compare against other heuristics which take into account state information. The heuristic is stated as follows. Class i tasks are mapped to machine j with probability

$$(2.6)\quad p_{i,j} = \frac{\delta^*_{i,j}\mu_{i,j}}{\lambda^*\alpha_i}.$$

The LP–Static heuristic maximizes system capacity in the long term, but may suffer from poor performance since it does not do any short-term shifting of workload among the machines. In Appendix A, it is proven that the LP–Static heuristic is guaranteed to stabilize a stabilizable system. However, the heuristic generally achieves poor performance. In Section 2.4, we introduce the Guided-LPAS heuristic and prove that it is guaranteed to stabilize a stabilizable system. The Guided-LPAS heuristic achieves competitive performance levels with the LPAS heuristic.

Table 2.4: Execution rates for System 2.$C2$

	Group					
Task	T	U	V	W	X	Y
1	16.7	24.8	24.2	29	25.6	48.3
2	30.4	48.3	77.7	83.6	135.9	144.9
3	18.9	24.2	48.3	45.8	72.5	72.5
4	3	3	7.6	7.6	8.3	8.7
5	1	1.1	3	2.9	3	3

2.3.2 Simulation Results

We use System 2.$C2$ which models a real cluster system [46] (for details, see He [35]) to compare the LPAS-related heuristics. System 2.$C2$ is a medium size system with 5 task classes and 30 machines. The machines are partitioned into 6 groups, machines within a group are identical. Groups T, U, V, W, X, and Y, consist of 2 machines, 6 machines, 7 machines, 7 machines, 4 machines, and 4 machines, respectively. The execution rates are shown in Table 2.4. The arrival rate vector is $\alpha = [204.10\ 68.87\ 77.63\ 5.01\ 10.43]$.

As done for System 2.$C1$ (Section 2.2.2), we solve the allocation LP corresponding to the following system:

$N = 5, M = 6, \alpha = [204.10\ 68.87\ 77.63\ 5.01\ 10.43]$, and

$$\mu = \begin{bmatrix} 33.4 & 148.8 & 169.4 & 203 & 102.4 & 193.2 \\ 60.8 & 289.8 & 543.9 & 585.2 & 543.6 & 579.6 \\ 37.8 & 145.2 & 338.1 & 320.6 & 290 & 290 \\ 6 & 18 & 53.2 & 53.2 & 33.2 & 34.8 \\ 2 & 6.6 & 21 & 20.3 & 12 & 12 \end{bmatrix}.$$

Solving the modified allocation LP gives $\lambda^* = 2.4242$ and

Table 2.5: The machine allocation matrix for System 2.$C2$

	Group					
Task	T	U	V	W	X	Y
1	1	1	0	0.5881	0	1
2	0	0	0	0	0.3071	0
3	0	0	0	0	0.6489	0
4	0	0	0	0.2009	0.0439	0
5	0	0	1	0.2111	0	0

$$\delta^* = \begin{bmatrix} 1 & 1 & 0 & 0.5881 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0.3071 & 0 \\ 0 & 0 & 0 & 0 & 0.6489 & 0 \\ 0 & 0 & 0 & 0.2009 & 0.0439 & 0 \\ 0 & 0 & 1 & 0.2111 & 0 & 0 \end{bmatrix}.$$

Thus, for System 2.$C2$, we use the δ^* matrix in Table 2.5. In this particular solution, machines that belong to the same group have identical values for $\delta^*_{i,j}$. Note that the number of nonzero elements in the δ^* matrix is 52. Using the LPAS heuristic, the discount of the average required state information for a mapping decision is 58%. On the other hand, using the LPAS–$2/k$ heuristic the discount is 94%.

Table 2.6 shows the simulation results for System 2.$C2$. As the table shows, the LPAS heuristic achieves the best results. The LPAS−$2/k$ heuristic has worse performance than that achieved by the LPAS heuristic, yet it uses less state information. The performance degradation is not large (and is significantly better than the LP–Static heuristic).

Table 2.6: Simulation results for System $2.C2$

LP–Static	$(24.25, 24.29)$
MCT	$(11.45, 11.46)$
LPAS	$(11.32, 11.33)$
LPAS$-2/k$	$(14.01, 14.02)$

2.4 The Guided-LPAS Heuristic

Consider the MCT heuristic. Stolyar [67] showed that it does not minimize system workload in heavy traffic. Sharifnia [63] showed that it may not stabilize the system even if the system can be stabilized. He attributed this to its greedy use of information resulting in assigning tasks to the "wrong" machines persistently and thus causing instability. An important question is: with the restrictions of the LPAS heuristic, is it true that the LPAS heuristic is guaranteed to stabilize a stabilizable system (*i.e.*, a system where the solution to the allocation LP is $\lambda^* > 1$)?

Even though our simulation experiments have failed to find a stabilizable system that is not stabilized by the LPAS heuristic, we are not able to prove the stability of the LPAS heuristic. This is because of the difficulty of finding an expression for the actual machine allocations achieved by the LPAS heuristic. However, we are confident of its stability as it avoids assigning tasks to the "wrong" machines by using task heterogeneity to provide foresight. Thus, it does not suffer from the root cause for the instability of the MCT heuristic. If one is still concerned about stability, we give the Guided-LPAS heuristic and give a proof for its stability.

The Guided-LPAS heuristic is guaranteed to stabilize a stabilizable system. It is a modification of the LPAS heuristic such that, over time, target (reference) execution capacities allocated for individual task classes on each machine are achieved. These targets are found from the solution of the allocation LP. In particular, the target execution capacity allocated by machine j for class i is $\frac{\delta^*_{i,j}\mu_{i,j}}{\lambda^*}$.

Let $\pi_{i,j}$ be the target mapping ratio of class i tasks to machine j such that the target execution capacity reference levels are achieved (*i.e.*, $\pi_{i,j} = \frac{\delta^*_{i,j}\mu_{i,j}}{\lambda^*\alpha_i}$). The Guided-LPAS heuristic uses the LPAS heuristic as long as the actual rate at which class i tasks are mapped to machine j is not too far from its target level $\pi_{i,j}$, or equivalently, the actual execution capacity levels are not far from their targets.

The Guided-LPAS heuristic can be stated as follows. Let $\alpha_{i,j}(t)$ denote the number of class i tasks assigned to machine j in $[0, t]$. Let $\alpha_i(t)$ denote the number of class i tasks that arrived during $[0, t]$. An arrival of a class i task at time t is mapped to a machine j for which: (i) the task's expected completion time is minimized, (ii) $\delta^*_{i,j} \neq 0$, and (iii) $\alpha_{i,j}(t^-) < \pi_{i,j}\alpha_i(t) + C_{i,j}\sqrt{t}$, where $C_{i,j}$ is a nonnegative but otherwise arbitrary constant. Note that since at any class i arrival time t, $\alpha_{i,j}(t^-) < \alpha_i(t)$ and $C_{i,j} \geq 0$, $j = 1, \ldots, M$, there is always at least one machine satisfying condition (iii), and therefore the heuristic is well defined.

The Guided-LPAS heuristic attempts to achieve the short-term advantages attained by the LPAS heuristic. However, it employs an oversight control that achieves target execution capacity reference levels in the long run. This ensures the stability of the heuristic while achieving good performance levels. The stability result for the Guided-LPAS heuristic is stated in the following theorem (the proof is provided in Appendix A):

Theorem 2.4.1 *The Guided-LPAS heuristic stabilizes a stabilizable system. More specifically, if the system is stabilizable and the mapper uses the Guided-LPAS heuristic, then the workload process ($\{W(t)\}$) converges to a steady-state distribution as $t \to \infty$.*

Our simulation experiments indicate that the oversight control mechanism is seldom used. For instance, consider the system defined in Section 2.1 to show the instability of the MCT heuristic. Setting $C_{i,j} = 1$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, and simulating the system under the Guided-LPAS heuristic (using the same assumptions as in Section 2.2.2), we observed that the number of times condition (iii) is violated was zero.

In Section 2.3, we introduced a variant of the LPAS heuristic which results in a further reduction of the state information required for mapping. We referred to this heuristic as the LPAS$-2/k$ heuristic. Here, we modify the LPAS$-2/k$ heuristic such that stability is guaranteed for stabilizable systems. The resulting heuristic is referred to as the Guided-LPAS$-2/k$ heuristic and is defined as follows. Let $T_i(t) = \{j \mid \delta^*_{i,j} \neq 0$ and $\alpha_{i,j}(t^-) < \pi_{i,j}\alpha_i(t) + C_{i,j}\sqrt{t}\}$. A class i arrival at time t is mapped to one of the two machines (j_1, j_2) chosen from $T_i(t)$ such that the arrival joins the machine with the minimum expected completion time. If $|T_i(t)| > 2$, machine j_1 is first chosen from $T_i(t)$ with probability $p_{j_1} = \frac{\delta^*_{i,j_1}\mu_{i,j_1}}{\sum_{j \in T_i(t)} \delta^*_{i,j}\mu_{i,j}}$. Then, machine j_2 is chosen from $T_i(t) \setminus \{j_1\}$ with probability $p_{j_2} = \frac{\delta^*_{i,j_2}\mu_{i,j_2}}{(\sum_{j \in T_i(t)} \delta^*_{i,j}\mu_{i,j}) - \delta^*_{i,j_1}\mu_{i,j_1}}$.

The following theorem states the stability result for the Guided-LPAS$-2/k$ heuristic (the proof is provided in Appendix A):

Theorem 2.4.2 *The Guided-LPAS$-2/k$ heuristic stabilizes a stabilizable system. More specifically, if the system is stabilizable and the mapper uses the Guided-LPAS$-2/k$ heuristic, then the workload process ($\{W(t)\}$) converges to a steady-state distribution as $t \to \infty$.*

2.5 Summary and Discussion

The main contribution of this chapter is the proposal of the LPAS mapping heuristic for heterogeneous computing systems. The LPAS heuristic utilizes the solution to an allocation LP in making mapping decisions. By solving an allocation LP, the LPAS heuristic provides an explicit method to compute the maximum capacity and to compute the allocation of machines to classes. This has the advantage of requiring dramatically less state information while at the same time achieving competitive performance levels. It does not suffer from the limitations of other mapping heuristics, namely the limited use of information about task heterogeneity (as in the case of the MCT heuristic) and the ad-hoc manner for setting parameters (as in the KPB heuristic). Furthermore, we have introduced two modifications to the LPAS heuristic. First, the LPAS$-2/k$ heuristic significantly reduces the state information required in mapping. Second, the Guided-LPAS heuristic is guaranteed to stabilize a stabilizable system.

A related open question is to analyze the robustness of the LPAS heuristic. Often, HC systems operate in an environment with a large degree of uncertainty (see Smith *et al.* [65]). In this context, robustness can be defined as the degree to which a system can function correctly in the presence of parameter values different from those assumed (Ali *et al.* [6]). A number of papers have studied robustness in HC systems, including Ali *et al.* [6], Mehta *et al.* [53], Shestak *et al.* [64], and Smith *et al.* [65]. We believe that the solution to the allocation LP is inherently robust with respect to errors in estimates of the parameters (the arrival and execution rates) and thus we expect the LPAS heuristic to have robustness advantages over other existing resource management policies for HC systems.

Chapter 3

Fault-Aware Scheduling Policies for Heterogeneous Desktop Grids

In this chapter, we introduce several scheduling policies for Desktop Grids. Section 3.1 describes the First-Come-First-Served policy which is used in major Desktop Grid schedulers. The G$c\mu$ policy and the LPAS_DG policy are described in Sections 3.2 and 3.3, respectively. In Section 3.4, we present the results obtained in our simulation experiments. In Section 3.5, we analyze the performance of the LPAS_DG policy using the McMaster Grid Scheduling Testing (MGST) framework. Section 3.6 concludes the chapter. Contents of this chapter appear in Al-Azzoni and Down [1, 4] and Kokaly *et al.* [42].

3.1 Current Policies

A scheduling policy that is applicable to our workload model is the classical First-Come-First-Served (FCFS) policy. FCFS is used in major Desktop Grid schedulers [27, 44]. An advantage of FCFS is that it does not require any information about task arrival rates or machine execution rates. However, as our simulations show, FCFS only performs well in systems with limited task heterogeneity and under moderate system loads. As the application tasks become more heterogeneous and the load increases, performance degrades rapidly.

3.2 The Gcμ Policy

The Gcμ policy is a variation of the generalized cμ rule (Gcμ) analyzed by Mandelbaum and Stolyar [52]. We consider the version of the Gcμ rule which asymptotically minimizes delay costs. The policy can be stated as follows: when a machine j requests a task, the scheduler assigns it the longest-waiting (head of the line) class i task such that $i \in \arg\max_i D_i(t)\mu'_{i,j}$, in which $D_i(t)$ is the waiting time (sojourn time) of the head of the line class i task at the time of making the scheduling decision t.

To the best of our knowledge, the Gcμ policy has never been suggested or used as a scheduling policy in Desktop Grids. The Gcμ policy aims at myopically maximizing the rate of decrease of the instantaneous delay cost. It has been proved that when the primitives α and μ satisfy certain conditions, the Gcμ policy minimizes both instantaneous and cumulative delay costs, asymptotically, over essentially all scheduling disciplines, preemptive or non-preemptive [52]. The optimality of the Gcμ policy is obtained under a heavy traffic assumption, in other words, optimality is achieved as the system load approaches 100 percent. When one backs off from the heavy traffic condition, we will see that there is room for making bad scheduling decisions, which in turn can significantly degrade performance.

Under moderate traffic conditions, the Gcμ rule could make more frequent bad scheduling decisions, especially in systems with highly heterogeneous execution rates. This results from the policy's greedy nature. Our LPAS_DG policy avoids this by preventing the assignment of particular task classes to inefficient machines.

Note that a scheduler using the Gcμ policy only requires information on the execution rates of the machines. Using this extra information, however, can result in achieving significant performance improvement over policies that do not use such information (*i.e.,* FCFS).

3.3 The LPAS_DG Policy

The LPAS_DG policy requires solving the following allocation LP (Andradóttir *et al.* [8]) at each machine availability/unavailability event, where the decision variables are λ and $\delta_{i,j}$ for $i = 1, \ldots, N$, $j = 1, \ldots, M$. The interpretation of the variables and constraints is identical to that of the allocation LP in Section 2.2.1.

$$\max \ \lambda$$

$$\text{s.t.} \ \sum_{j=1}^{M} \delta_{i,j}\mu'_{i,j} \geq \lambda\alpha_i, \qquad \text{for all } i = 1, \ldots, N, \tag{3.1}$$

$$\sum_{i=1}^{N} \delta_{i,j} \leq a_j, \qquad \text{for all } j = 1, \ldots, M, \tag{3.2}$$

$$\delta_{i,j} \geq 0, \qquad \text{for all } i = 1, \ldots, N, \text{ and } j = 1, \ldots, M. \tag{3.3}$$

Whenever a machine becomes available or unavailable, the scheduler solves the allocation LP to find $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. If a machine j becomes unavailable, then $a_j = 0$. In this case, $\delta^*_{i,j} = 0$ for $i = 1, \ldots, N$. On the other hand, if a machine j becomes available, a_j is equal to the predicted CPU availability for machine j during its next expected machine availability period (CPU availability prediction techniques are discussed in Section 1.4.2). Solving the allocation LP at each availability/non-availability event represents how the LPAS_DG policy adapts to the dynamics of machine availability. Constraint (3.2) enforces the condition that the allocation of machine j should not exceed its CPU availability. The use of a_j represents how the LPAS_DG policy adapts to the dynamics of CPU availability.

The value λ^* can be interpreted as follows. Consider an event in which a machine becomes available or unavailable. Let λ^* and $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, be an optimal solution to the allocation LP corresponding to the system just after the occurrence of the event. Consider the system that only consists of the available subset of the M machines. Then, the value λ^* can also be interpreted as the maximum capacity of this partial system.

The LPAS_DG policy is defined as follows. When a machine j requests a task, the scheduler assigns machine j the longest-waiting (head of the line) class i task such that $\mu_{i,j}\delta^*_{i,j} > 0$ and $i \in \arg\max_i \mu_{i,j} D_i(t)$, where $D_i(t)$ is defined in Section 3.2. Note that $\mu_{i,j}$ represents the effective execution rate for class i tasks at machine j ($\mu_{i,j} = a_j \mu'_{i,j}$ for $i = 1, \ldots, N$, $j = 1, \ldots, M$).

Consider a system with two machines and two classes of tasks ($M = 2$, $N = 2$). The arrival and execution rates are as follows:

$$\alpha = \begin{bmatrix} 1 & 1.5 \end{bmatrix} \text{ and } \mu = \begin{bmatrix} 9 & 5 \\ 2 & 1 \end{bmatrix}.$$

Assume that all machines are dedicated (*i.e.*, $a_j = 1$, for all $j = 1, \ldots, M$). Solving the allocation LP gives $\lambda^* = 1.7647$ and

$$\delta^* = \begin{bmatrix} 0 & 0.3529 \\ 1 & 0.6471 \end{bmatrix}.$$

Thus, when machine 1 requests a task, the scheduler only assigns it a class 2 task. Machine 2 can be assigned tasks belonging to any class. Although the fastest rate is for machine 1 at class 1, machine 1 is never assigned a class 1 task. Note that machine 1 is twice as fast as machine 2 on class 2 tasks and note that $\frac{\mu_{1,1}}{\mu_{2,1}} < \frac{\mu_{1,2}}{\mu_{2,2}}$.

Now assume that machine 1 is fully dedicated and machine 2 is available only 10% of the time (*i.e.*, $a_1 = 1$ and $a_2 = 0.1$). Solving the new allocation LP gives $\lambda^* = 1.2258$ and

$$\delta^* = \begin{bmatrix} 0.0806 & 0.1 \\ 0.9194 & 0 \end{bmatrix}.$$

In this case, machine 1 is assigned tasks from any class, but machine 2 is only assigned class 1 tasks. Note that machine 1 is 20 times as fast as machine 2 on class 2 and thus the LPAS_DG policy avoids assigning a class 2 task to machine 2.

To show how the LPAS_DG policy adapts to machine failures, consider the following system ($M = 4$, $N = 3$). The arrival and execution rates are as follows:

$$\alpha = \begin{bmatrix} 3 & 5 & 4 \end{bmatrix} \text{and } \mu = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 1 & 20 & 3.7 & 5.9 \\ 1 & 20 & 7.1 & 2.7 \end{bmatrix}.$$

Assume that all machines are dedicated (*i.e.*, $a_j = 1$, for all $j = 1, \ldots, M$). Solving the allocation LP gives $\lambda^* = 2.0513$ and

$$\delta^* = \begin{bmatrix} 1 & 0.0769 & 1 & 1 \\ 0 & 0.5128 & 0 & 0 \\ 0 & 0.4103 & 0 & 0 \end{bmatrix}.$$

Note that machine 1 is never assigned tasks belonging to class 2 or class 3. While machine 2 may be assigned tasks from any class, machines 3 and 4 are only assigned class 1 tasks.

Now, assume that machine 2 fails. Solving the new allocation LP gives $\lambda^* = 1.0306$ and

$$\delta^* = \begin{bmatrix} 1 & 0 & 0.4194 & 0.1266 \\ 0 & 0 & 0 & 0.8734 \\ 0 & 0 & 0.5806 & 0 \end{bmatrix}.$$

In this case, in addition to class 1 tasks, machine 3 is assigned class 3 tasks and machine 4 is assigned class 2 tasks.

Ideally, the number of zero elements in the δ^* matrix should be $NM+1-N-M$. If the number of zero elements is greater, the LPAS_DG policy would be significantly restricted in shifting workload between machines resulting in performance degradation. Also, if the number of zero elements is very small, the LPAS_DG policy resembles more closely the G$c\mu$ policy. In fact, if the δ^* matrix contains no zeros at all, then the LPAS_DG policy reduces to the G$c\mu$ policy. Throughout the chapter and unless otherwise stated, we use an optimal solution in which the δ^* matrix contains exactly $NM + 1 - N - M$ zeros. Such a solution always exists (see Proposition 2.2.1).

3.4 Simulation Results

We use simulation to compare the performance of the scheduling policies. The task arrivals are modeled by independent Poisson processes, each with rate α_i, $i = 1, \ldots, N$. The execution times are exponentially distributed with rates $\mu'_{i,j}$, where $1/\mu'_{i,j}$ represents the mean execution time of a task of class i at machine j, $i = 1, \ldots, N$, $j = 1, \ldots, M$. Unless otherwise stated, it is assumed that machine fault times and availability times are exponentially distributed. A machine fault (availability) time represents a time interval during which the machine is unavailable (available).

There are several performance metrics that can be used to compare the performance of the scheduling policies [9, 44]. We use the long-run average task completion time $\bar{W}$, as a metric for performance comparison. A task completion time is defined as the time elapsing between the submission of the task and the completion of its execution, including resubmission times. For each simulation experiment, we also show the average task completion time for class i tasks, $\bar{W}_i$, for all $i = 1, \ldots, N$.

In this section, we study several systems. Each simulation experiment models a particular system under different assumptions on machine and CPU availabilities. Each experiment simulates the execution of the corresponding system for 20,000 time-units. Each experiment is repeated 30 times. For every case, we give $\bar{W}$, the improvement (Δ) over the G$c\mu$ policy, and $\bar{W}_i$, $i = 1, \ldots, N$. For $\bar{W}$, we also give the accuracy of the confidence interval defined as the ratio of the half width of the interval over the mean value (all statistics are at 95% confidence level). A negative improvement means a policy is being outperformed by the G$c\mu$ policy. Note that we do not give

Table 3.1: Execution rates for System 3.A

Task	Group					
	T	U	V	W	X	Y
1	2	2	2	2	2	2
2	1	20	3.7	7.1	2.4	8.7
3	1	20	9.4	3.7	7.3	2.7
4	1	20	2.8	5.9	4.4	6.3

performance results for the FCFS policy when it results in either an unstable system or one in which performance is several orders of magnitude worse than the Gcμ policy.

Consider the following system. System 3.A is a medium-size system with 4 task classes and 30 machines. The machines are partitioned into 6 groups, with machines within a group being identical. Thus, if two machines are in the same group, then they have the same execution rates. Groups T and U consist of 3 machines each, while groups V, W, X, and Y consist of 6 machines each. For the systems discussed in this section, the machines are ordered with the machines of group T first, group U second, etc. Thus, for example, in System 3.A, the machine $j = 7$ belongs to group V and the machine $j = 30$ belongs to group Y. The execution rates are shown in Table 3.1. Using this partition, we have all machines being homogeneous to class 1 tasks; 10 percent of machines are slow for most arrivals, 10 percent of machines are fast for most arrivals and the majority of machines (the remaining 80 percent) have high task and machine heterogeneity.

For System 3.A, Table 3.2 shows the simulation results under two different arrival streams: (i) $\alpha^1 = [11.25\ 22.5\ 36\ 63]$, and (ii) $\alpha^2 = [17.5\ 35\ 56\ 98]$. The arrival rates α^1 result in a lightly loaded system while those in α^2 lead to a heavily loaded system.

The following are the simulation scenarios for arrival rates α^1:

1. There are no machine failures, and $a_j = 1$ for all $j = 1, \ldots, M$.

2. Each machine fails at the rate 0.02 per time-unit and the mean fault time is two

Table 3.2: Simulation results for System 3.A

Under Arrival Rates α^1																	
	FCFS						$Gc\mu$					LPAS_DG					
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	1.65 $\pm$6.90%	**−617.39%**	1.93	1.63	1.63	1.62	0.23 $\pm$0.03%	0.54	0.20	0.19	0.20	0.15 $\pm$0.04%	**34.78%**	0.51	0.13	0.12	0.11
2							0.23 $\pm$0.04%	0.55	0.20	0.19	0.20	0.15 $\pm$0.06%	**34.72%**	0.51	0.13	0.12	0.11
3							0.25 $\pm$0.16%	0.64	0.21	0.19	0.23	0.18 $\pm$0.26%	**28%**	0.56	0.17	0.15	0.13
4							0.25 $\pm$0.04%	0.62	0.21	0.20	0.23	0.18 $\pm$0.05%	**28%**	0.58	0.16	0.13	0.13
5							0.31 $\pm$0.33%	0.80	0.26	0.22	0.28	0.24 $\pm$0.48%	**22.58%**	0.68	0.23	0.19	0.18
Under Arrival Rates α^2																	
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1							0.40 $\pm$0.44%	1.11	0.33	0.30	0.37	0.32 $\pm$0.40%	**20%**	1.01	0.25	0.27	0.25
2							0.48 $\pm$0.56%	1.30	0.38	0.34	0.44	0.38 $\pm$0.47%	**20.83%**	1.09	0.28	0.31	0.32
3							0.81 $\pm$1.21%	2.20	0.64	0.57	0.77	0.62 $\pm$0.93%	**23.46%**	1.88	0.40	0.48	0.56

time-units. Machines are fully dedicated when they are available *i.e.*, $a_j = 1$ for all $j = 1, \ldots, M$.

3. Each machine fails at the rate 0.05 per time-unit and the mean fault time is four time-units. Machines are fully dedicated when they are available *i.e.*, $a_j = 1$ for all $j = 1, \ldots, M$. Failures in this case are more common than the previous case.

4. Each machine fails at the rate 0.02 per time-unit and the mean fault time is two time-units. CPU availabilities are given by:

$$a_j = \begin{cases} 0.5 & \text{if } j = 13, 19, 25, \\ 0.75 & \text{if } j = 1, 4, 14, 26, \\ 1 & \text{otherwise.} \end{cases}$$

5. Each machine fails at the rate 0.05 per time-unit and the mean fault time is four time-units. CPU availabilities are the same as in the previous case.

The following are the simulation scenarios for arrival rates α^2:

1. There are no machine failures, and $a_j = 1$ for all $j = 1, \ldots, M$.

2. Each machine fails at the rate 0.01 per time-unit and the mean fault time is one time-unit. Machines are fully dedicated when they are available *i.e.*, $a_j = 1$ for all $j = 1, \ldots, M$.

3. Each machine fails at the rate 0.01 per time-unit and the mean fault time is one time-unit. CPU availabilities are given by:

$$a_j = \begin{cases} 0.75 & \text{if } j = 13, 19, \\ 0.85 & \text{if } j = 14, 25, \\ 1 & \text{otherwise.} \end{cases}$$

The simulation results above suggest that using the LPAS_DG policy results in improved performance over the Gcμ policy. Also, using the FCFS policy for System 3.A results in severe performance degradation.

3.4.1 Task and Machine Heterogeneity

Systems 3.B through 3.E model different kinds of system heterogeneity. The simulation results for these systems are presented in Tables 3.3, 3.4, 3.5, and 3.6, respectively. We model each system under two different sets of arrival rates: α^1 and α^2. The arrival rates α^1 result in a lightly loaded system compared to a heavily loaded system under arrival rates α^2. The following are the simulation scenarios for arrival rates α^1:

1. There are no machine failures, and $a_j = 1$ for all $j = 1, \ldots, M$.
2. Each machine fails at the rate 0.05 per time-unit and the mean fault time is four time-units. Machines are fully dedicated when they are available *i.e.*, $a_j = 1$ for all $j = 1, \ldots, M$.

The following are the simulation scenarios for arrival rates α^2:

1. There are no machine failures, and $a_j = 1$ for all $j = 1, \ldots, M$.
2. Each machine fails at the rate 0.02 per time-unit and the mean fault time is two time-units. Machines are fully dedicated when they are available *i.e.*, $a_j = 1$ for all $j = 1, \ldots, M$.

For Systems 3.B through 3.E, $M = 28$ and $N = 4$. The machines are partitioned into 7 groups (labeled T through Z). Each group consists of 4 machines and machines within a group are identical.

System 3.B models a HiHi system. The arrival rate vectors are $\alpha^1 = [50\ 48\ 50\ 48]$ and $\alpha^2 = [62.5\ 60\ 62.5\ 60]$. The execution rates are shown in Table 3.7.

System 3.C models a LoHi system. The arrival rate vectors are $\alpha^1 = [30\ 30\ 24\ 24]$ and $\alpha^2 = [40\ 40\ 32\ 32]$. The execution rates are shown in Table 3.8.

Table 3.3: Simulation results for System 3.B

Under Arrival Rates α^1										
	Gcμ					LPAS_DG				
Case	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	0.22 ±0.04%	0.21	0.23	0.20	0.23	0.13 ±0.04%	**40.91%**	0.12	0.14	0.13	0.12
2	0.37 ±0.70%	0.35	0.42	0.35	0.36	0.28 ±1.10%	**24.32%**	0.27	0.32	0.27	0.26
Under Arrival Rates α^2											
Case	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	0.28 ±0.14%	0.27	0.32	0.27	0.27	0.22 ±0.30%	**21.43%**	0.24	0.21	0.27	0.19
2	0.45 ±0.85%	0.42	0.54	0.42	0.41	0.39 ±0.79%	**17.78%**	0.37	0.45	0.40	0.32

Table 3.4: Simulation results for System 3.C

Under Arrival Rates α^1																	
	FCFS						Gcμ					LPAS_DG					
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	0.21 ±0.02%	**0%**	0.21	0.21	0.20	0.21	0.21 ±0.02%	0.21	0.21	0.20	0.21	0.22 ±0.03%	**-4.76%**	0.21	0.11	0.29	0.29
2	0.27 ±0.78%	**0%**	0.27	0.28	0.27	0.28	0.27 ±0.76%	0.26	0.27	0.27	0.27	0.31 ±0.73%	**-14.81%**	0.30	0.21	0.38	0.38
Under Arrival Rates α^2																	
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	0.27 ±0.33%	**-3.85%**	0.27	0.27	0.26	0.27	0.26 ±0.23%	0.25	0.25	0.27	0.26	0.32 ±0.23%	**-23.08%**	0.31	0.25	0.38	0.37
2	0.65 ±2.77%	**-38.30%**	0.65	0.65	0.64	0.65	0.47 ±1.21%	0.46	0.45	0.50	0.47	0.52 ±0.97%	**-10.64%**	0.49	0.44	0.56	0.59

Table 3.5: Simulation results for System 3.D

Under Arrival Rates α^1															
	FCFS						Gcμ					LPAS_DG					
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	0.21 ±0.06%	**0%**	0.48	0.26	0.18	0.11	0.21 ±0.06%	0.49	0.26	0.17	0.10	0.23 ±0.08%	**-9.52%**	0.47	0.26	0.21	0.13
2	1.46 ±4.64%	**-204.17%**	1.74	1.50	1.42	1.35	0.48 ±1.09%	1.07	0.59	0.41	0.24	0.54 ±1.17%	**-12.5%**	1.04	0.64	0.46	0.35
Under Arrival Rates α^2																	
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	1.04 ±3.38%	**-205.88%**	1.31	1.08	1.00	0.93	0.34 ±0.44%	0.75	0.42	0.29	0.17	0.54 ±1.07%	**-58.82%**	1.04	0.64	0.45	0.35
2							0.75 ±1.88%	1.65	0.92	0.63	0.38	0.77 ±1.64%	**-2.67%**	1.54	0.88	0.73	0.43

Table 3.6: Simulation results for System 3.E

Under Arrival Rates α^1																	
	FCFS						Gcμ					LPAS_DG					
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	0.20 ±0.03%	**0%**	0.20	0.20	0.21	0.20	0.20 ±0.04%	0.20	0.20	0.21	0.20	0.22 ±0.05%	**-10%**	0.22	0.21	0.23	0.21
2	0.28 ±0.58%	**-3.70%**	0.28	0.28	0.29	0.28	0.27 ±0.52%	0.27	0.26	0.28	0.27	0.33 ±0.49%	**-22.22%**	0.33	0.32	0.34	0.32
Under Arrival Rates α^2																	
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	0.46 ±1.15%	**-35.29%**	0.45	0.45	0.46	0.46	0.34 ±0.53%	0.34	0.34	0.35	0.34	0.45 ±0.48%	**-32.35%**	0.49	0.43	0.52	0.40
2							1.18 ±2.93%	1.17	1.17	1.20	1.17	1.12 ±2.44%	**5.08%**	1.05	1.01	1.22	1.21

Table 3.7: Execution rates for System 3.B

	Group						
Task	T	U	V	W	X	Y	Z
1	4.5	2	9.5	6.2	10.25	2.25	3.95
2	6.2	4.5	6	2	4.2	5.9	10.25
3	9.5	6.5	4	10	5.9	2.25	3.95
4	2.25	10	2	3.95	1.75	10	1.75

Table 3.8: Execution rates for System 3.C

	Group						
Task	T	U	V	W	X	Y	Z
1	2.2	7	10.25	1	5.7	0.5	12
2	1.95	7.05	9.78	0.95	5.65	0.56	11.85
3	2	7.25	10.02	0.98	5.75	0.67	11.8
4	2.05	6.75	9.99	1.02	5.82	0.49	12.05

Table 3.9: Execution rates for System 3.D

Task	Group						
	T	U	V	W	X	Y	Z
1	2	2.5	2.25	2	2.2	1.75	2.25
2	4.5	4	4.2	4	3.8	3.9	3.95
3	6	6.2	6.25	6	5.75	5.9	6.05
4	10	10.25	10.5	9.5	10.25	10.25	10

Table 3.10: Execution rates for System 3.E

Task	Group						
	T	U	V	W	X	Y	Z
1	5	5.05	4.95	4.98	4.7	5.2	5.25
2	5.25	5.09	4.9	4.92	5	5.13	5.14
3	4.45	5	4.9	4.45	4.9	5	5.1
4	5.02	4.95	5	5.02	5.25	4.75	5

System 3.D models a HiLo system. The arrival rate vectors are $\alpha^1 = [14\ 28\ 35\ 35]$ and $\alpha^2 = [17\ 34\ 42.5\ 42.5]$. The execution rates are shown in Table 3.9.

System 3.E models a LoLo system. The arrival rate vectors are $\alpha^1 = [24\ 27\ 21\ 30]$ and $\alpha^2 = [32\ 36\ 28\ 40]$. The execution rates are shown in Table 3.10.

The results in Tables 3.4 and 3.6 indicate that the FCFS policy achieves acceptable performance in lightly loaded systems with low task heterogeneity. The FCFS policy achieves poor performance and even results in unstable systems as the level of task heterogeneity increases or as the system load increases. This suggests that FCFS will not be able to support the same level of throughput as our two proposed policies. While the LPAS_DG policy achieves very competitive performance to that of the Gcμ policy, its performance is generally superior only in highly heterogeneous and highly loaded systems.

3.4.2 Other Distributions

To test the sensitivity of the performance of the scheduling policies to the distributional assumptions, Table 3.11 shows the simulation results for System 3.A under arrival rates α^1 using different distributions for machine execution times, fault times, and availability times. With respect to machine failure rates and CPU availabilities, cases 1, 2, and 3 in Table 3.11 correspond to cases 1, 3, and 5 in Table 3.2, respectively.

For the execution times, in addition to the exponential distribution, two other distributions are used to study lower and higher variances than the exponential case: the first is a constant execution time of size $\frac{1}{\mu_{i,j}}$ for machine j executing class i tasks, and the second is a hyper-exponential distribution with mean $\frac{1}{\mu_{i,j}}$ for the execution times and twice the variance as the exponential case. For the machine availability and fault times, a hyper-exponential distribution is used. In agreement with the data provided in [56], the squared coefficient of variation for the hyper-exponential distribution is set to five.

The simulation results above indicate that the relative performance of the policies is not affected by the distributions for machine execution times, fault times, and availability times. Furthermore, the performance of the FCFS policy depends on the level of task heterogeneity, and in systems with highly heterogeneous tasks, the policy performs poorly regardless of the underlying distributions.

3.4.3 Large Systems

System 3.F is a large system with $M = 3000$ machines and $N = 4$ classes. The system is constructed using 100 multiples of System 3.A. Table 3.12 shows the simulation results for System 3.F under arrival rates $\alpha = 100 \times \alpha^1$, where α^1 represents the first set of arrival rates used in simulating System 3.A (see Table 3.2).

The following are the simulation scenarios for arrival rates α:

1. Each machine fails at the rate 0.05 per time-unit and the mean fault time is four

Table 3.11: Simulation results for System 3.A involving other distributions

Exponential Execution Time Distribution																	
	FCFS						Gcμ					LPAS_DG					
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	1.63 ±5.58%	**-608.70%**	1.91	1.61	1.61	1.61	0.23 ±0.04%	0.54	0.20	0.19	0.20	0.15 ±0.04%	**34.78%**	0.51	0.13	0.12	0.11
2							0.25 ±0.56%	0.64	0.21	0.20	0.23	0.18 ±0.68%	**28%**	0.56	0.17	0.15	0.13
3							0.32 ±1.47%	0.83	0.27	0.23	0.30	0.22 ±2.46%	**31.25%**	0.70	0.20	0.17	0.18
Hyper-exponential Execution Time Distribution																	
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	7.13 ±10.41%	**-3000%**	7.41	7.11	7.11	7.11	0.23 ±0.05%	0.54	0.20	0.19	0.20	0.15 ±0.05%	**34.78%**	0.51	0.13	0.12	0.11
2							0.26 ±0.52%	0.65	0.22	0.20	0.24	0.19 ±0.82%	**26.92%**	0.56	0.19	0.17	0.14
3							0.34 ±1.16%	0.86	0.28	0.24	0.31	0.26 ±1.39%	**23.53%**	0.71	0.27	0.22	0.20
Deterministic Execution Time Distribution																	
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	0.94 ±2.52%	**-327.27%**	1.21	0.91	0.91	0.91	0.22 ±0.02%	0.53	0.19	0.19	0.20	0.15 ±0.02%	**31.82%**	0.51	0.12	0.11	0.11
2							0.24 ±0.63%	0.61	0.20	0.18	0.21	0.17 ±0.67%	**29.17%**	0.55	0.15	0.14	0.13
3							0.29 ±1.07%	0.77	0.24	0.21	0.27	0.22 ±1.5%	**24.14%**	0.69	0.20	0.17	0.18

Table 3.12: Simulation results for System 3.F

	Gcμ					LPAS_DG					
Case	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
1	0.19 ±0.0052%	0.50	0.16	0.16	0.16	0.14 ±0.0078%	**26.32%**	0.50	0.11	0.11	0.11
2	0.19 ±0.0093%	0.55	0.16	0.15	0.16	0.15 ±0.0075%	**21.05%**	0.55	0.13	0.12	0.11

time-units. Machines are fully dedicated when they are available *i.e.*, $a_j = 1$ for all $j = 1, \ldots, M$.

2. Each machine fails at the rate 0.05 per time-unit and the mean fault time is four time-units. Since System 3.F is constructed using 100 multiples of System 3.A, the CPU availabilities for each multiple are given as those for System 3.A under case 4 (refer to the simulated cases for System 3.A under arrival rates α^1).

As Table 3.12 shows, the LPAS_DG policy achieves the best results for System 3.F. The FCFS policy results in significant performance degradation, although the system is not heavily loaded under arrival rates α. These results indicate that, even for large systems, the relative performance of the policies depends on the heterogeneity of the system as well as its load.

3.4.4 The Value of Information on CPU Availabilities

Consider System A. Assume that each machine fails at the rate 0.05 per time-unit and the mean fault time is four time-units. CPU availabilities are given by:

$$a_j = \begin{cases} 0.05 & \text{if } j = 4, 5, 7, 13, 19, 20, 25, \\ 1 & \text{otherwise.} \end{cases}$$

We simulate the system under arrival rates $\alpha = 0.75 \times \alpha^1 = [8.4375\ 16.875\ 27\ 47.25]$, where α^1 represents the first set of arrival rates used in simulating System A. We consider two cases. In the first case, the policy does not use estimated CPU availabilities (*i.e.*, the policy assumes that $a_j = 1$, for all $j = 1, \ldots, M$). In the second case, the policy uses the estimated CPU availabilities. Our simulation experiments indicate that the LPAS_DG policy which incorporates information on CPU availabilities results in $\Delta = 20.51\%$ while the LPAS_DG policy which does not use this information results in $\Delta = -156.41\%$. These results show that the LPAS_DG policy effectively exploits knowledge on CPU availabilities. Furthermore, the LPAS_DG policy may perform poorly when these estimates are not available. In such cases, the use of the G$c\mu$ policy is recommended.

3.4.5 Realistic Architectures

To simulate more realistic scenarios, we use the data reported in [9] and Canonico [15] which was collected by running benchmarking tools on an actual system. We refer to this system as System 3.G.

In [9], the authors define the nominal computing power of a machine as a real number whose value is directly proportional to its speed. Thus, a machine with a nominal computing power of 2 is twice as fast as a machine with a nominal computing power of 1. It is found that, for System 3.G, there are three different values for the nominal computing power of machines, namely $\{1, 1.125, 1.4375\}$.

Since we consider the problem of scheduling multiple applications on Desktop Grids, we define $P_{i,j}$ as the nominal computing power of machine j on class i tasks. Thus, a machine j with $P_{i,j} = 2$ is twice as fast as a machine j' with $P_{i,j'} = 1$ on class i tasks. In this manner, we can describe systems in which a machine is fast on some applications but slow on others.

As in [9], the CPU availability is described by a discrete-time Markov chain whose parameters are computed using a network monitoring and forecasting system. A new

Table 3.13: A copy of Table 4.14 in [15]

Hostname	P	shape	scale	P_{aa}	P_{ab}	P_{ac}	P_{ba}	P_{bb}	P_{bc}	P_{ca}	P_{cb}	P_{cc}
bird	1	0.568664	477343	0.998	0.001	0.001	0.015	0.970	0.015	0.000	0.0197	0.9803
blind	1.125	0.729215	856663	0	0	0	0	0.9814	0.0186	0	0.0098	0.9902
booboo	1	0.570816	619497	0	0	0	0	0.9904	0.0096	0	0.0133	0.9867
chocolate	1.4375	0.662438	610445	0	0	0	0	0.9958	0.0042	0	0.1376	0.8624
hobbes	1.125	0.560362	199690	0.9969	0.0029	0.0002	0.0270	0.9674	0.0056	0.0005	0.0123	0.9872
joplin	1	0.969769	1271536	0.9987	0.0011	0.00003	0.0028	0.9946	0.0024	0.0002	0.0134	0.9863
kenny	1.125	0.729823	350024	0	0	0	0	0.9914	0.0086	0	0.0152	0.9848
marge	1.4375	0.677637	373307	0.9982	0.0017	0.0001	0.0097	0.9735	0.0168	0.0005	0.0449	0.9546
marvin	1	0.928094	753368	0	0	0	0	0.9795	0.0205	0	0.0378	0.9622
miles	1.125	0.570816	619497	0	0	0	0	0.9933	0.0067	0	0.0503	0.9497
nat	1.4375	0.607016	405233	0	0	0	0	0.9946	0.0054	0	0.0352	0.9648
popeye	1	0.616905	228117	0	0	0	0	0.9889	0.0111	0	0.0214	0.9786
rocky	1.125	0.537631	178959	0	0	0	0	0.9964	0.0036	0	0.0004	0.9996
scooby	1.4375	0.68684	248058	0.9982	0.0016	0.0002	0.0093	0.9815	0.0092	0.0005	0.0169	0.9826
taz	1.4375	0.556867	243961	0	0	0	0	0.9916	0.0084	0	0.025	0.9750

value for the CPU availability is computed every 10 seconds of simulated time. The chain contains three states: a, b, and c, corresponding to the CPU availability of 100%, 50%, and 33%, respectively. Let P_{xy} denote the one-step transition probability of moving from state x to state y $(x, y \in \{a, b, c\})$. The actual values for each machine's transition probabilities are reported in Table 4.14 in [15] (the table is reproduced in Table 3.13).

To find the steady-state probability P_x of being at a state x, we solve the following set of equations:

$$P_a = P_a P_{aa} + P_b P_{ba} + P_c P_{ca}$$

$$P_b = P_a P_{ab} + P_b P_{bb} + P_c P_{cb}$$

$$P_c = P_a P_{ac} + P_b P_{bc} + P_c P_{cc}$$

together with the normalizing equation $P_a + P_b + P_c = 1$.

For the LPAS_DG policy, we compute a_j as the steady-state CPU availability for each machine j from the corresponding Markov chain:

$$a_j = 1.00P_a + 0.50P_b + 0.33P_c$$

This is justified for the model of System 3.G since the mean execution time for a given task is much larger than the average time spent in a particular state of the Markov chain.

To model machine availability, we use a Weibull distribution. The density and distribution functions for a Weibull distribution are given by (v denotes the machine availability time, $v \in (0, \infty)$) [56]:

$$f_W(v) = \alpha\beta^{-\alpha}v^{\alpha-1}e^{-(v/\beta)^{\alpha}},$$

$$F_W(v) = 1 - e^{-(v/\beta)^{\alpha}},$$

respectively. The parameter α is called the shape parameter, and β is called the scale parameter. The actual values for the Weibull parameters depend on the particular machine. For System 3.G, these parameters are provided in Table 4.14 in [15]. As in [9], the fault time of a machine is set to a constant 120 time-units.

We simulate two configurations based on System 3.G (3.$G1$ and 3.$G2$). Both systems consist of $M = 300$ machines. We simulate the execution of each system for two billion time-units. We group the machines into 15 groups. Each group consists of 20 machines identical in terms of the Markov chain describing CPU availability and the parameters for the Weibull distribution. Each group has the same parameters as those of one of the 15 machines of System 3.G listed in Table 4.14 in [15].

In System 3.$G1$, we assume that the machines of a group are identical in terms of their nominal computing powers. Each group has the same nominal computing power as one of the 15 machines of System 3.G. Furthermore, we assume that the nominal computing power of a machine depends only on the machine and is independent of the class of tasks being executed. Thus, if a machine j belongs to a group G and the nominal computing power for the group is P_G, then $P_{i,j} = P_G$, for all $i = 1, \ldots, N$. Thus, a fast machine is fast on all applications. System 3.$G1$ represents a system which is mainly used to execute a single application.

In System 3.$G2$, we assume that each machine has a nominal computing power (on class i tasks) $P_{i,j}$ randomly chosen from $\{1, 1.125, 1.4375\}$ with equal probabilities. Thus, a machine can be fast executing some applications while, at the same time, slow executing other applications. System 3.$G2$ represents a system which is mainly used to execute multiple applications with inherent heterogeneity.

Finally, we assume that there are $N = 4$ classes (or applications). The authors in [9] define $BaseTime$ as the mean execution time of a task submitted to a machine with a nominal computing power of 1. Thus, each class consists of tasks with the same value for $BaseTime$ (for class i, we denote it by $BaseTime_i$). We assume that $BaseTime_i$ = 8750, 17500, 35000, 50000, for $i = 1, \ldots, 4$, respectively. This

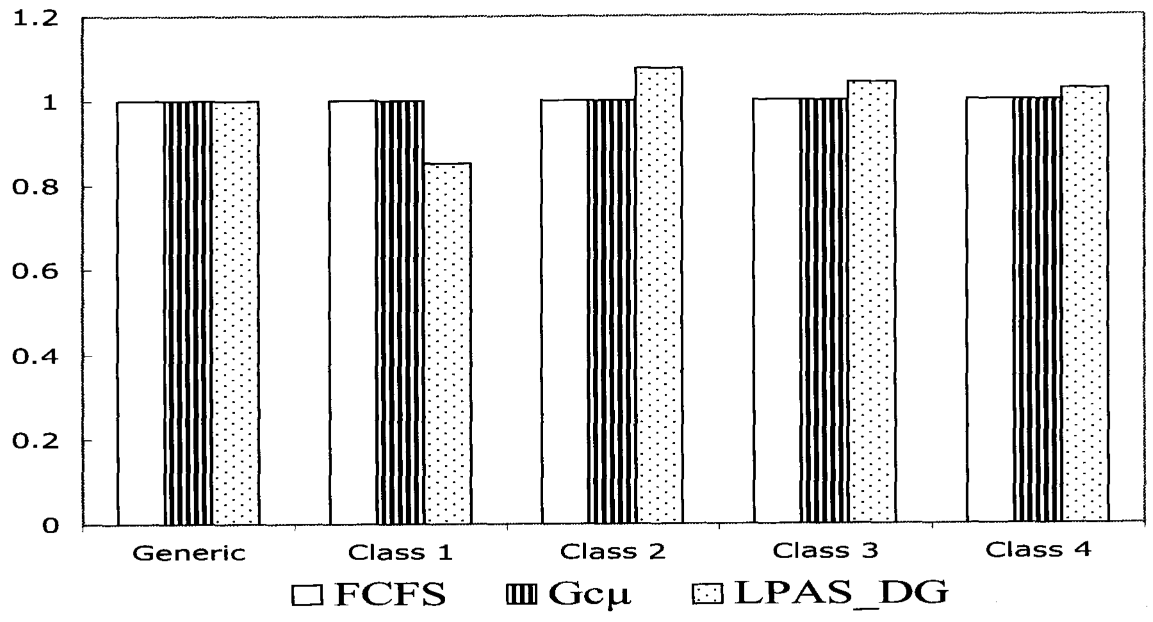

Figure 3.1: Relative average task completion times: System 3.$G1$ under arrival rates α^1

information is enough to generate the matrix μ'. The mean nominal execution time for a class i task at machine j can be computed as $BaseTime_i \times 1/P_{i,j}$.

Figures 3.1 and 3.2 show simulation results for Systems 3.$G1$ and 3.$G2$ under arrival rates $\alpha^1 = [0.00457 \;\; 0.00229 \; 0.00114 \; 0.00080]$. In this section, we normalize the results with respect to the Gcμ policy and note that the accuracy of the generated confidence intervals is 0.1% or less. These results indicate that the FCFS policy achieves acceptable performance in systems with low task heterogeneity, such as System 3.$G1$. However, as the level of task heterogeneity increases (*e.g.* System 3.$G2$), FCFS results in performance degradation which gets worse as the load increases. For instance, Figure 3.3 shows results for System 3.$G2$ under higher load ($\alpha^2 = [0.00495 \; 0.00110 \; 0.00214 \; 0.00135]$). In this case, FCFS results in an unstable system. Both the Gcμ and the LPAS_DG policies result in significant performance improvement. The LPAS_DG policy is generally superior in highly heterogeneous and highly loaded systems.

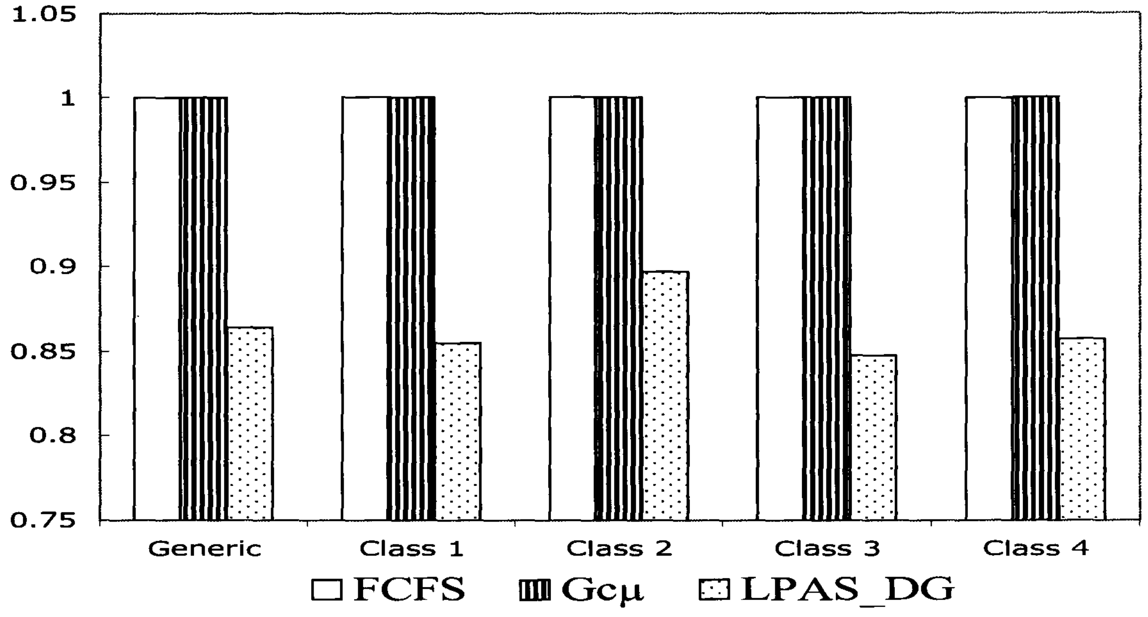

Figure 3.2: Relative average task completion times: System 3.$G2$ under arrival rates α^1

3.4.6 Robust Modifications

Throughout the previous simulation experiments, we have assumed that the LPAS_DG policy uses an optimal solution in which the δ^* matrix contains exactly $NM + 1 - N - M$ zeros. Such a restriction reduces the number of machines that can execute each task class. In some cases, especially in systems with low task heterogeneity, this may result in performance degradation. Furthermore, as observed in Section 3.5, this causes the LPAS_DG policy to be less robust against potential parameter estimation errors and other sources of errors.

In this section, we modify the LPAS_DG policy by eliminating such a restriction. However, we avoid the use of optimal solutions having no zero elements in the δ^* matrix, since in this case the LPAS_DG policy reduces to the Gcμ policy. To do so, we use the optimal solutions provided by the barrier optimization routine (CPXbaropt) of ILOG CPLEX [37]. By alleviating such a restriction on the number of zero elements in the δ^* matrix, the LPAS_DG policy becomes less aggressive in its exclusion of

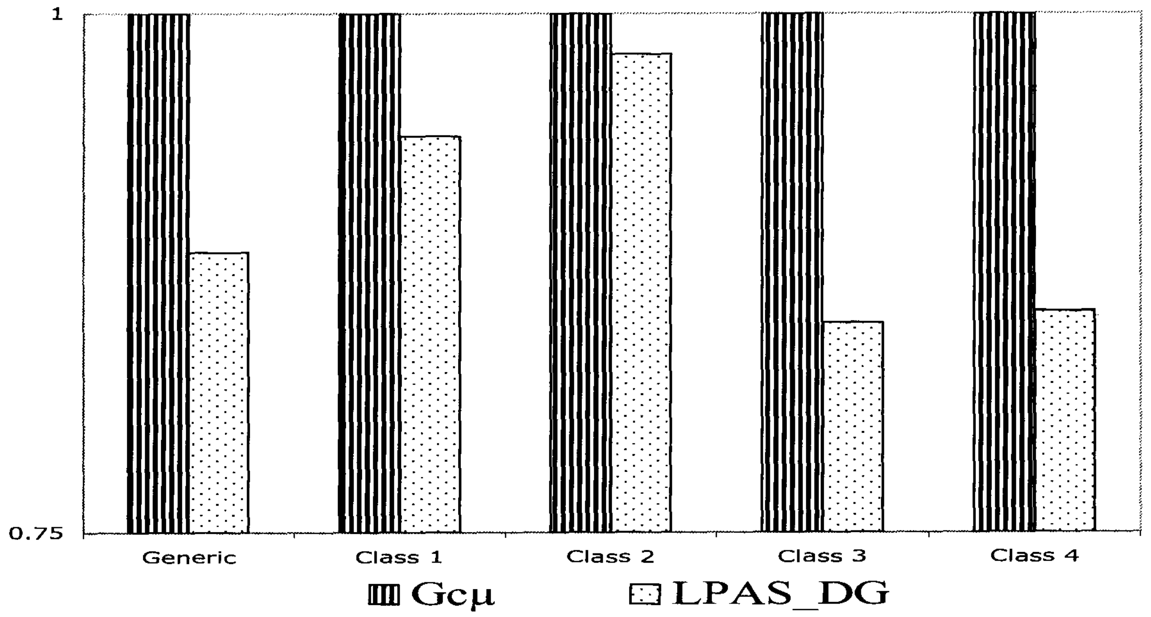

Figure 3.3: Relative average task completion times: System 3.$G2$ under arrival rates α^2

machines for certain task classes. As our simulations show, this results in improved performance and increased robustness.

Table 3.14 shows simulation results for the systems of different heterogeneity considered in Section 3.4.1. These results show that the modified LPAS_DG policy results in significant performance improvement over the unmodified version. Furthermore, performance is improved with respect to the Gcμ policy: the degradation becomes less in the case of the LoHi System (System 3.C) and a positive improvement results in the case of the HiLo System (System 3.D).

In the following experiment, we compare the unmodified LPAS_DG policy against the modified version with respect to their robustness against CPU availability estimates. Consider the following system (System 3.H). The system has identical machines as System 3.A. We simulate the system under arrival rates α^1 (see System 3.A). Each machine fails at the rate 0.02 per time-unit and the mean fault time is two time-units. CPU availabilities are given by:

$$a_j = \begin{cases} 0.25 & \text{if } j = 6, 10, 16, 22, \\ 0.5 & \text{if } j = 5, 9, 14, 15, 21, 27, \\ 0.75 & \text{if } j = 4, 7, 8, 13, 19, 20, 25, 26, \\ 1 & \text{otherwise.} \end{cases}$$

Using an approach similar to Iosup *et al.* [39] and Zhang and Inoguchi [73], we assess the impact of inaccuracy under the assumption of null overall inaccuracy [39]. Under this assumption, while any individual estimate may be inaccurate, the (overall) average estimation inaccuracy is 0. Define I to be the maximum inaccuracy whose value ranges from 0% (perfect information) to 100% (high inaccuracy). When a machine j becomes available, let a'_j denote the estimated CPU availability for machine j used by the LPAS_DG policy in solving the allocation LP. In our simulations, a'_j is obtained using the following relation: $a'_j = a_j \times (1 + E)$, where E is sampled from the uniform distribution $[-I, +I]$ and a_j is the actual CPU availability for machine j. If $a_j \times (1 + E) > 1$, we set a'_j to 1; and similarly, if $a_j \times (1 + E) < 0$, we set a'_j to 0.

Figure 3.4 compares the two versions of the LPAS_DG policy in terms of their performance improvement with respect to the Gcμ policy. The figure shows that the modified version is more robust against CPU availability estimates, while the unmodified version may result in negative improvement under larger values of I. This is due to the aggressiveness of the policy in minimizing the number of machines to execute each task class.

3.5 Implementation

In this section, we use the McMaster Grid Scheduling Testing (MGST) framework to analyze the performance of the LPAS_DG policy. MGST, the first performance testing framework for Desktop Grids, was developed by researchers at McMaster

Table 3.14: Simulation results using the modified LPAS_DG policy

Under Arrival Rates α^1							Under Arrival Rates α^2						
Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$	Case	$\bar{W}$	Δ	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}_4$
System 3.B													
1	0.12 ±0.04%	**45.45%**	0.11	0.13	0.12	0.11	1	0.17 ±0.18%	**39.29%**	0.18	0.18	0.18	0.16
2	0.25 ±0.95%	**35.14%**	0.24	0.27	0.25	0.24	2	0.31 ±1.14%	**26.19%**	0.30	0.35	0.32	0.28
System 3.C													
1	0.21 ±0.03%	**0%**	0.21	0.11	0.27	0.27	1	0.31 ±0.19%	**-19.23%**	0.35	0.25	0.33	0.31
2	0.29 ±0.59%	**-7.41%**	0.29	0.20	0.36	0.33	2	0.49 ±0.93%	**-4.26%**	0.54	0.44	0.49	0.50
System 3.D													
1	0.23 ±0.08%	**-9.52%**	0.45	0.25	0.24	0.11	1	0.42 ±0.49%	**-23.53%**	0.74	0.44	0.53	0.16
2	0.46 ±1.08%	**4.17%**	0.93	0.55	0.39	0.27	2	0.69 ±1.63%	**8%**	1.50	0.74	0.67	0.34
System 3.E													
1	0.21 ±0.04%	**-5%**	0.21	0.22	0.21	0.20	1	0.40 ±0.44%	**-17.65%**	0.42	0.52	0.40	0.28
2	0.29 ±0.40%	**-7.41%**	0.29	0.29	0.29	0.29	2	0.95 ±3.13%	**19.49%**	0.95	1.00	0.95	0.90

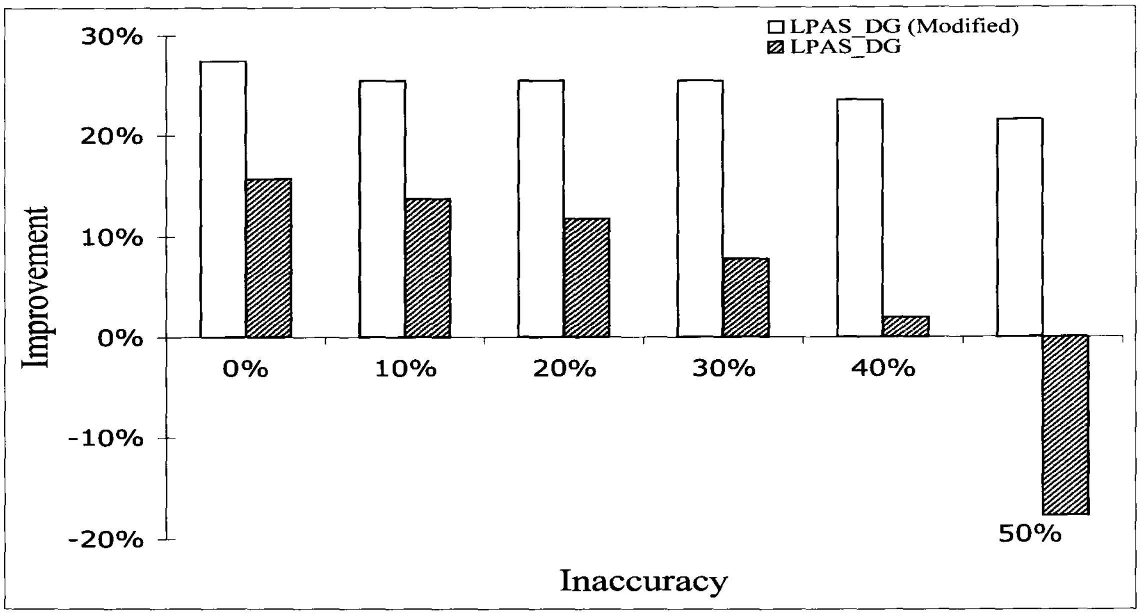

Figure 3.4: Performance improvements under different values for the maximum inaccuracy I

University (Kokaly *et al.* [42]). MGST simplifies and automates the process of realistic performance testing over a distributed testbed complementing the use of simulation. We use the results from the MGST deployment to make several recommendations for the practical application of the LPAS_DG policy.

3.5.1 Experimental Results

We used MGST to analyze the performance of the LPAS_DG policy under realistic conditions. We tested the scheme on several systems. The systems used were Intel (dual-core 2.0 GHz) and PowerPC (single-core 2.0 GHz) based Macintosh computers. The systems were located on the same network. Each test was conducted two times, once using the simulation tool used in Section 3.4 and once with MGST. The metric used in the simulations and experiments is the average response time, including average communication delay for the MGST experiments. The communication delay is the difference between the time a task is sent to be executed and the time it begins execution. This delay occurs mainly due to network communication delays, but it could also be caused by the software layer responsible for the distribution and execution of the tasks.

The experiments were conducted on four categories of systems depending on machine and task heterogeneity:

– High task heterogeneity and high machine heterogeneity (HiHi).

– High task heterogeneity and low machine heterogeneity (HiLo).

– Low task heterogeneity and high machine heterogeneity (LoHi).

– Low task heterogeneity and low machine heterogeneity (LoLo).

Two to four experiments were conducted on each category. In some experiments failures were enabled meaning that machines can fail while executing tasks. Machines were in some experiments fully dedicated ($a_j = 1$ for all j), where their full resources were used exclusively by the desktop grid. In other experiments only a percentage of the resources were available for the grid. We will use the following acronyms to express these properties in the experiments: FE, FD, MFD, MPD for failures enabled, failures disabled, machine fully dedicated and machines partially dedicated respectively.

The experiments in the HiHi category were conducted on 6 machines and 4 classes of tasks. Machines 1 to 6 have the same execution rates as those of Groups T to Y in Table 3.1, respectively. The arrival rates of the task classes were: $\alpha = [2.25\ 4.50\ 7.20\ 12.60]$.

The average response time for each class of tasks and the over all average response time are shown in Table 3.15. The simulation results in this and all of the following tables are at a 95% confidence interval.

In the experiments MPD/FD and MPD/FE machines 4, 5 and 6 had availability $a_j = 0.5$. The remaining machines were fully dedicated ($a_j = 1$). In the MFD/FE and MPD/FE experiments each machine failed at the rate 0.05 per time-unit and the mean fault time was 2 time-units. The periods were exponentially distributed.

Table 3.15: Results of experiment on HiHi setting

	MFD/FD		MPD/FD		MFD/FE		MPD/FE	
Class	Sim	MGST	Sim	MGST	Sim	MGST	Sim	MGST
1	(0.56, 0.57)	0.58	(0.97, 0.98)	0.99	(0.61, 0.61)	0.61	(1.10, 1.11))	1.03
2	(0.33, 0.34)	0.34	(0.22, 0.22)	0.37	(0.35, 0.35)	0.45	(1.10, 1.11)	0.50
3	(0.18, 0.18)	0.22	(0.27, 0.27)	0.29	(0.19, 0.20)	0.20	(0.32, 0.32)	0.46
4	(0.11, 0.11)	0.17	(0.16, 0.16)	0.43	(0.13, 0.13)	0.15	(0.26, 0.27)	1.36
Overall	(0.20, 0.21)	0.24	(0.27, 0.27)	0.42	(0.23, 0.23)	0.25	(0.35, 0.36)	0.94

In the experiments MPD/FD and MPD/FE the actual performance of the LPAS_DG policy was much worse than the simulation had predicted. This is discussed in detail in Section 3.5.2.

The LoHi setting was constructed from 21 machines and 4 task classes. There were seven groups of machines with each group having 3 machines. Members of the same group had the same execution rates. Machines in group 1 are machines 1, 2 and 3, machines in group 2 are machines 4, 5 and 6, etc. Groups 1 to 7 have the same execution rates as those of Groups T to Z in Table 3.8, respectively. The arrival rates of the task classes were: $\alpha = [22.5\ 22.5\ 18.0\ 18.0]$.

The average response time for each class of tasks and the over all average response time are shown in Table 3.16.

In the MPD/FD experiment machines 4, 11 and 15 had availability $a_j = 0.5$. Machines 7, 14 and 18 had availability $a_j = 0.75$. The remaining machines were fully dedicated ($a_j = 1$).

The average response times of the MGST experiment were slightly higher due to the fact that actual execution rates were somewhat slower. This is discussed in Section 3.5.2.

The HiLo setting was constructed from 21 machines and 4 task classes. The machines were divided into seven groups in the same way machines in the setting

Table 3.16: Results of experiment on LoHi setting

	MFD/FD		MPD/FD	
Class	Sim	MGST	Sim	MGST
1	(0.22, 0.22)	0.31	(0.24, 0.24)	0.36
2	(0.12, 0.12)	0.22	(0.13, 0.13)	0.26
3	(0.30, 0.30)	0.37	(0.37, 0.37)	0.44
4	(0.29, 0.29)	0.35	(0.35, 0.35)	0.47
Overall	(0.22, 0.22)	0.31	(0.26, 0.27)	0.37

LoHi were divided. Groups 1 to 7 have the same execution rates as those of Groups T to Z in Table 3.9, respectively. The arrival rates of the task classes were: $\alpha =$ [10.50 21.00 26.25 26.25].

The average response time for each class of tasks and the over all average response time are shown in Table 3.17. The availabilities of machines were as in the LoHi setting.

Table 3.17: Results of experiment on HiLo setting

	MFD/FD		MPD/FD	
Class	Sim	MGST	Sim	MGST
1	(0.49, 0.49)	0.50	(0.79, 0.80)	1.22
2	(0.28, 0.28)	0.31	(0.42, 0.42)	0.77
3	(0.24, 0.24)	0.32	(0.27, 0.27)	0.53
4	(0.14, 0.14)	0.35	(0.19, 0.19)	0.73
Overall	(0.25, 0.25)	0.35	(0.35, 0.35)	0.74

Compared to simulation, the LPAS_DG policy performed poorly in the MGST experiment. The reason is that the ideal overall load on the machines was fairly high (86.4%), but the different sources of errors and overhead caused the actual load to be close to 100%. The sources of errors are higher overall arrival rates, over estimation for execution rates and communication overhead coupled with the scheduling delay. See Section 3.5.2 for more details.

The LoLo setting was constructed from 21 machines and 4 task classes. The machines were divided into seven groups in the same way machines in the setting LoHi were divided. Groups 1 to 7 have the same execution rates as those of Groups T to Z in Table 3.10, respectively. The arrival rates of the task classes were: $\alpha = [18.00\ 20.25\ 15.75\ 22.50]$.

The average response time for each class of tasks and the over all response time are shown in Table 3.18. This experiment included machine failures. In the MFD/FE and MPD/FE experiments each machine failed at the rate 0.05 per time-unit and the mean fault time was 2 time-units. The periods were exponentially distributed. The availabilities of machines were as in the LoHi setting.

Table 3.18: Results of experiment on LoLo setting

	MFD/FD		MPD/FD		MFD/FE		MPD/FE	
Class	Sim	MGST	Sim	MGST	Sim	MGST	Sim	MGST
1	(0.25, 0.25)	0.27	(0.28, 0.28)	0.39	(0.25, 0.25)	0.35	(0.31, 0.31)	0.52
2	(0.23, 0.23)	0.28	(0.30, 0.30)	0.39	(0.24, 0.24)	0.34	(0.32, 0.32)	0.63
3	(0.23, 0.23)	0.28	(0.27, 0.27)	0.35	(0.24, 0.24)	0.33	(0.32, 0.32)	0.57
4	(0.21, 0.22)	0.25	(0.32, 0.32)	0.36	(0.24, 0.24)	0.29	(0.34, 0.34)	0.52
Overall	(0.23, 0.23)	0.27	(0.30, 0.30)	0.37	(0.24, 0.24)	0.33	(0.32, 0.32)	0.56

The response times in the results of our experiment were significantly higher than the simulation results. The reason behind this is the high load coupled with failures and over estimation of the execution rates (the assumed execution rates were higher than the actual ones in this experiment). See Section 3.5.2.

3.5.2 Analysis and Recommendations

The LPAS_DG policy was implemented for the first time in MGST. Here we give a few remarks regarding the implementation of this policy.

The LPAS_DG policy makes decisions based on the matrix δ^*, which is produced by solving an allocation LP. The δ^* matrix depends on the values of a_j. As a result, it is suggested in Section 3.3 that a new δ^* matrix must be produced at every availability/unavailability event.

Since the matrix δ^* depends on a_j, if the machines' a_j varies between availability and unavailability events, we think that δ^* should be updated every time any a_j changes. This solution is expensive to implement because it is very hard to notify the scheduler of every change to every a_j. In addition, this will require solving the allocation LP frequently, which is also expensive and may raise a scalability problem (the scheduler could become the bottleneck). To solve this issue, we assumed a time resolution T_{system}. Here, the values of a_j are sent to the scheduler periodically, and it solves the allocation LP after receiving the updated values of a_j. The determination of an optimal update period is open to research. We believe that this modification is necessary to make the LPAS_DG policy scalable.

In some experiments the performance of the scheduling schemes differed from the simulation results due to the machines experiencing unexpectedly high loads. The different sources of error that can occur in a real system can significantly raise the load, even potentially causing instability in the system. These errors can be caused by:

1. **The measured arrival rates being larger than that assumed.**

2. **Overestimation of execution rates.**

3. **Overhead caused by communication and scheduling delays.** Assume that a machine announces its availability at time t_1, the scheduler learns of the availability of this machine at time t_2 and consequently performs the scheduling

and chooses a task to send at time t_3. The machine then receives the task and starts the execution at time t_4. At time t_5 the machine finishes executing the task but only at time t_6 does the scheduler learn that the task is done, obtaining the results at t_7. In the model, the execution time is considered to be $t_5 - t_4$, but in the actual implementation, there is an overhead of $(t_4 - t_1) + (t_7 - t_5)$. This overhead affects the load if the overhead is significant with respect to $t_5 - t_4$.

4. **Machine failures.** Although machine failures can be incorporated in the workload models, they can still increase the effective load due to the fact that it takes time for the scheduler to realize that a machine is down. This time is wasted and effectively increases the load. For example, when using the LPAS_DG policy, suppose that machine 3 is the only machine executing tasks from class 1, and the execution time is 5 minutes. If machine 3 fails when executing a particular task and the "time-out" parameter was set to 3 times (i.e. 3 times the estimated execution time should elapse before considering the task "timed out"), then the scheduler will not consider machine 3 down until 15 minutes have elapsed from the moment that the task was sent. These 15 minutes are essentially lost, with arriving tasks from class 1 accumulating in the queue at the scheduler within that time.

If any or all of the above factors cause a significant increase in the load, the performance of the scheduling scheme will deteriorate. Note that these factors were only discovered upon deploying the LPAS_DG policy on MGST. They were not discovered in simulations.

The LPAS_DG policy suffered in some cases in the experiments from the above factors due to the aggressive nature of this policy in minimizing the number of machines to execute each task class. This results in exclusivity of machines for certain task classes. When one class can be executed by a small number of machines, then the performance depends only on these machines, so the effect of the factors mentioned above is magnified. Contrast this with FCFS, where if a machine under performs, the effect is less obvious since this under performing machine can get help from other (potentially over performing) machines. Finally, the scheduling delay can contribute to the time needed to execute tasks, effectively raising the load on machines for all policies.

As a result of the MGST experiments, we propose the following suggestions to improve robustness of the LPAS_DG policy:

1. **Arrival rates estimation improvement.** Since the LPAS_DG policy depends on solving an allocation LP and that in turn depends on values that include arrival rates of task classes, estimates should be as accurate as possible. To do so, we propose that the actual arrival rates should be monitored (a feature that MGST provides), and check the values against the estimated values every specific time ($T_{arrival\_rate}$) and resolve the LP if one of the actual values differs from its estimate by a specific threshold percentage ($Th_{arrival\_rate}$) that depends on the load and the task class. $T_{arrival\_rate}$ could be a specific time period or a number of task arrivals from a class (e.g. 10 tasks). We believe that this solution is not computationally costly, since the checking operation requires $O(N)$ time and $O(N)$ space. We expect the number of task classes (N) to be relatively small, so there should be no scaling issues. An alternative solution is to over estimate the arrival rates of classes, however, caution must be taken to guarantee that the system is theoretically stable.

2. **Avoiding execution rates overestimation.** We propose that every execution rate entry (for a specific machine for a specific task class) is modified then checked (against the estimated peer) whenever a task is done, then the LP is resolved if that entry differs from the estimated one by a specific threshold percentage ($Th_{execution\_rate}$) that depends on the load and the task class. This solution requires $O(NM)$ space and $O(1)$ time. Alternatively, the execution rates can be assumed slower than they are estimated to be in a manner that guarantees that they can never be over estimated (how to do this is not clear).

Caution must be taken to assure that the system is theoretically stable.

3. **Lessen the effect of communication and scheduling delays.** Let $p_{i,j}$ be an estimate of the value

   $$\frac{1/\mu_{i,j}}{1/\mu_{i,j} + \tau_j}$$

   where τ_j is the communication and scheduling delay for machine j.

   In the example mentioned when discussing the sources of errors (see point 3), $p_{i,j}$ would be

   $$\frac{t_5 - t_4}{t_7 - t_1}$$

   where the times are for the particular choice of i and j.

   We propose that all execution rates must be multiplied by $p_{i,j}$ before resolving the LP to take this effect into consideration.

4. **Lessen the machine failure effect.** We propose choosing a low value for the time out, which will result in allowing the scheduler to quickly detect machine failures. The downside of this approach is that the scheduler might consider a machine failed when it is not (in particular, when availability rapidly decreases for a machine).

3.6 Summary

A distinct feature for this work is the proposal of fault-aware policies that take into consideration the heterogeneity of Desktop Grids. We have proposed to use the Gcμ policy for Desktop Grids when information on the machine execution rates is available. When task arrival rates and CPU availabilities are available, we have developed the LPAS_DG policy which utilizes the solution to an allocation LP. Both policies perform much better than FCFS, especially for applications with high task heterogeneity. There are some cases for which the G$c\mu$ policy is recommended over the LPAS_DG policy: i) when the applications have limited task heterogeneity, ii) when the system has limited machine heterogeneity, or iii) when there is a high level of inaccuracy in the estimation of task arrival rates, machine execution rates, or CPU availabilities. Otherwise, the performance of the LPAS_DG policy is significantly better, especially in highly heterogeneous systems.

Chapter 4

Decentralized Load Balancing for Heterogeneous Grids

In this chapter, we introduce a new decentralized load balancing for grids. Section 4.1 describes several related policies. The LPAS_dec policy is described in Section 4.2. In Section 4.3, we present the results obtained in our simulation experiments. Section 4.4 concludes the chapter. Contents of this chapter appear in Al-Azzoni and Down [2].

4.1 Current Policies

As discussed earlier, the MCT (minimum completion time) policy assigns an arriving task to the machine that has the earliest expected completion time. Several authors have suggested decentralized load balancing policies that are based on the MCT policy, *e.g.*, the LBA (Load Balancing on Arrival) policy in [62] and the IDP (Instantaneous Distribution Policy) in [50]. When a task arrives to a machine, the machine contacts all machines in the system to determine the machine with the earliest expected completion time.

There are several limitations to such policies. First, when a task arrives to a machine, the machine requires full state information. As explained in Section 1.3, policies that require full state information may suffer from performance degradation due to the effect of outdated information. Furthermore, the policy suffers from a significant information exchange overhead. In particular, for each arriving task to a machine, the machine needs to send a request message to all machines in the system who in turn need to send back a reply message containing the expected completion time information. Thus, a total of $2 \times (M - 1)$ message exchanges are needed for every arriving task. Further discussion on the MCT policy is provided in Section 2.1.

An advantage of the MCT policy is that a machine does not require any information about the task arrival rates or machine execution rates of other machines. Thus, only the expected completion times need to be exchanged between machines. It is for this reason that the MCT policy is suited for systems where predicting these rates is not possible or severely inaccurate.

In order to address the limitations of the MCT policy, we look at a decentralized version of the KPB policy (see Section 2.1). With respect to a machine j, let $S_i^{k,j}$ be the set of the $\lfloor kM/100 \rfloor$ machines that have the smallest expected execution time for class i tasks. When a task of class i arrives to machine j, the machine assigns the task to the machine in the subset $S_i^{k,j}$ that has the earliest expected completion time. Define $\overline{k} = \lfloor kM/100 \rfloor$.

The KPB policy requires knowledge on machine execution rates while the MCT policy does not. We use the following mechanism for exchanging information on machine execution rates. When a task of class i arrives to a machine j', the machine sends request messages to the machines $j \in S_i^{k,j'}$. In each request message, machine j' includes its local execution rates ($\mu_{i,j'}$, $i = 1, \ldots, N$). Upon receiving the request messages, each of the contacted machines replies with a message including the corresponding expected completion time as well as the local execution rates. Thus, at the end, machine j' and the machines $j \in S_i^{k,j'}$ update their local state information with the corresponding execution rates.

4.2 The LPAS_dec Policy

The LPAS_dec policy is similar to the KPB policy in that only a subset of machines need to be considered for each class, however, the determination of this subset requires solving the following LP [7], where the decision variables are λ and $\delta_{i,j}$ for $i = 1, \ldots, N$, $j = 1, \ldots, M$. The interpretation of the variables and constraints is identical to that of the allocation LP in Section 2.2.1.

$$\max \ \lambda$$

(4.1)
$$\text{s.t.} \ \sum_{j=1}^{M} \delta_{i,j}\mu_{i,j} \geq \lambda\alpha_i, \qquad \text{for all } i = 1, \ldots, N,$$

(4.2)
$$\sum_{i=1}^{N} \delta_{i,j} \leq 1, \qquad \text{for all } j = 1, \ldots, M,$$

(4.3)
$$\delta_{i,j} \geq 0, \qquad \text{for all } i = 1, \ldots, N, \ \text{and } j = 1, \ldots, M.$$

The LPAS_dec policy can be stated as follows. Each machine j' solves a local version (using local data) of the allocation LP to find $\{\delta^*_{i,j}\}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. When a new task of class i arrives to a machine j', let $S_i^{j'}$ denote the set of machines whose corresponding $\delta^*_{i,j}$ at machine j' is not zero. Machine j' assigns the task to the machine $j \in S_i^{j'}$ that has the earliest expected completion time among the subset of machines $S_i^{j'}$. Again, ties are broken arbitrarily.

The LPAS_dec policy requires knowledge on both arrival and execution rates. We use the following mechanism for information exchange. When a task of class i arrives to a machine j', the machine sends request messages to the machines $j \in S_i^{j'}$. In each request message, machine j' includes its local arrival and execution rates ($\alpha_{i,j'}$, $\mu_{i,j'}$, $i = 1, \ldots, N$). Upon receiving the request messages, each of the contacted machines replies with a message including the corresponding expected completion time as well as the local arrival and execution rates. Thus, at the end, machine j' and the machines $j \in S_i^{j'}$ update their local state information with the corresponding arrival and execution rates.

Under ideal conditions when full state information is available, all the machines solve the same allocation LP and thus use the same δ^* matrix, achieving the maximum capacity. However, in practice, at any given time, one or more of the machines may have different views of the state of the system (here, the state of the system refers to the arrival and execution rates). Thus, they solve different allocation LPs and the resulting δ^* matrices are different. However, as our simulation experiments indicate, the information exchange mechanism of the LPAS_dec policy is effective in its state update and thus the machines tend to quickly have the same view of the system. Furthermore, since the LPAS_dec policy does not use the actual values for $\{\delta^*\}$ (it only uses information on what entries are nonzero), and since these LPs are inherently robust with respect to the arrival and execution rates, the resulting δ^* matrices tend to be similar with respect to the positions of the zero and nonzero entries. Thus, performance would not be significantly deteriorated when the observed system state is a little different amongst the machines. This also explains the observed robustness of the LPAS_dec policy against parameter estimation errors (see Section 4.3.2).

Consider a system with two machines and two classes of tasks ($M = 2$, $N = 2$). Assume initially that α and μ are known by both machines:

$$\alpha = \begin{bmatrix} 1 & 1.45 \\ 1 & 1.45 \end{bmatrix}, \text{ and } \mu = \begin{bmatrix} 9 & 5 \\ 2 & 1 \end{bmatrix}.$$

Solving the allocation LP gives

$$\delta^* = \begin{bmatrix} 0 & 0.5 \\ 1 & 0.5 \end{bmatrix}.$$

Thus, all arriving tasks that belong to class 1 are assigned to machine 2. At the times of their arrivals, tasks that belong to class 2 are assigned to the machine, either machine 1 or 2, that has the earliest expected completion time.

Now, assume that $\alpha_{1,2}$ becomes 0.5. Thus, machine 2 solves a new allocation LP to obtain:

$$\delta^* = \begin{bmatrix} 0 & 0.3273 \\ 1 & 0.6727 \end{bmatrix}.$$

Even though machines 1 and 2 use different δ^* matrices until the next state update, they are equivalent in terms of the locations of the zero and nonzero entries. Thus, machine 1 still uses an allocation matrix that is equivalent in effect to the allocation matrix which maximizes the system capacity.

Ideally, the number of zero elements in the δ^* matrix should be $NM + 1 - N - M$. If the number of zero elements is greater, the LPAS_dec policy would be significantly restricted in shifting workload between machines resulting in performance degradation. Furthermore, in this case, our information exchange mechanism becomes less effective in its state update. Also, solutions that result in degenerate cases should be avoided. For example, if the δ^* matrix contains no zeros at all, then the LPAS_dec policy reduces to the MCT policy. Throughout the chapter, we use an optimal solution in which the δ^* matrix contains exactly $NM + 1 - N - M$ zeros.

4.3 Simulation Results

We use simulation to compare the performance of several load balancing policies including the LPAS_dec policy. In Section 4.3.1, we simulate an artificial system with high heterogeneity levels to show the impact of heterogeneity on performance. Then, in Section 4.3.3, we show the results of simulating a realistic grid.

The task arrivals are modeled by independent Poisson processes, each with rate $\alpha_{i,j}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. The execution times are exponentially distributed with rates $\mu_{i,j}$, where $1/\mu_{i,j}$ represents the mean execution time of a task of class i at machine j, $i = 1, \ldots, N$, $j = 1, \ldots, M$.

We use the long-run average task completion time $\bar{W}$ (defined in Section 3.4), as a metric for performance comparison. For each simulation experiment, we also show the average task completion time for class i tasks, $\bar{W}_i$, for all $i = 1, \ldots, N$. Another metric we also show is the total number of message exchanges, X. With respect to a given policy, a larger value for X indicates more overhead is involved in state information exchange.

In this section, we define several systems. Each simulation experiment models a particular system, characterized by the values of M, N, $\alpha_{i,j}$, and $\mu_{i,j}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$. Each experiment is repeated 30 times. For every case, we give $\bar{W}$, $\bar{W}_i$, $i = 1, \ldots, N$, and X. For $\bar{W}$, we also give the accuracy of the confidence interval defined as the ratio of the half width of the interval over the mean value (all statistics are at 95% confidence level).

4.3.1 Task and Machine Heterogeneity

System 4.A has $M = 7$ machines and $N = 4$ classes. Define α^1 and μ^1 as follows:

$$\alpha^1 = \begin{bmatrix} 2 & 1.5 & 1.75 & 1 & 3 & 1.9 & 1.35 \\ 1.35 & 1.5 & 2.4 & 1.55 & 2.9 & 0.75 & 1.55 \\ 4 & 2.75 & 1 & 1.35 & 1.5 & 0.9 & 1 \\ 2 & 1.75 & 2 & 1.5 & 2.25 & 1.75 & 0.75 \end{bmatrix}$$

and

$$\mu^1 = \begin{bmatrix} 4.5 & 2 & 9.5 & 6.2 & 10.25 & 2.25 & 3.95 \\ 6.2 & 4.5 & 6 & 2 & 4.2 & 5.9 & 10.25 \\ 9.5 & 6.5 & 4 & 10 & 5.9 & 2.25 & 3.95 \\ 2.25 & 10 & 2 & 3.95 & 1.75 & 10 & 1.75 \end{bmatrix}.$$

Initially, the arrival and execution rates are given by $\alpha = \alpha^1$ and $\mu = \mu^1$. The rates only change at regular rate-change events. At every rate-change event, only a single rate from $\alpha_{i,j}$ or $\mu_{i,j}$, $i = 1, \ldots, N$, $j = 1, \ldots, M$, changes randomly with

Table 4.1: Simulation results for System 4.A

Policy	$\bar{W}$	$\bar{W}_1$	$\bar{W}_2$	$\bar{W}_3$	$\bar{W}4$	Δ
MCT	**3.41** $\pm$13.31%	3.40	3.40	3.40	3.43	**0%**
KPB $\bar{k} = 2$	**0.37** $\pm$4.04%	0.28	0.58	0.40	0.20	**66.67%**
KPB $\bar{k} = 3$	**0.24** $\pm$0.59%	0.20	0.25	0.27	0.26	**50.02%**
LPAS_dec	**0.22** $\pm$0.36%	0.23	0.19	0.24	0.21	**65.19%**

equal probabilities. Time intervals between the rate-change events are exponentially distributed with mean 1/0.035 time-units. For a change in α, $\alpha_{i,j}$ is set to $\alpha^1_{i,j}$, $1.1\alpha^1_{i,j}$, or $1.2\alpha^1_{i,j}$ with equal probabilities. For a change in μ, $\mu_{i,j}$ is set to $\mu^1_{i,j}$, $1.05\mu^1_{i,j}$, or $1.15\mu^1_{i,j}$ with equal probabilities. Thus, the system experiences different loads with the lowest average load of 77.59% (when $\alpha = \alpha^1$ and $\mu = 1.15\mu^1$) and the highest average load of 89.23% (when $\alpha = 1.2\alpha^1$ and $\mu = \mu^1$).

Table 4.1 shows simulation results for System 4.A. We simulate the execution of the system for 200,000 time-units. In the last column of the table, we define Δ as the improvement in the total number of message exchanges (X) with respect to the MCT policy. For X, the accuracy of the confidence intervals is less than 0.1%.

The MCT policy performs much worse than the other policies. In general, the MCT policy achieves poor performance and even results in unstable systems when the system is highly loaded and there is high task heterogeneity and high machine heterogeneity. Using the KPB policy, performance is dramatically improved with respect to the MCT policy. However, finding an appropriate value for $\bar{k}$ is problematic. Furthermore, depending on the value of $\bar{k}$, there is a tradeoff between the achieved performance ($\bar{W}$) and the overhead of message exchanges (X). The LPAS_dec policy achieves the best performance for System A even though there is a dramatic decrease in the total number of message exchanges. It results in values for Δ comparable with those of the KPB policy with $\overline{k} = 2$, while achieving an improvement of 40% in the average task completion time.

4.3.2 Robustness

We assess the impact of inaccuracy under the assumption of null overall inaccuracy (see Section 3.4.6). Consider an actual arrival rate $\alpha_{i,j}$ for class i tasks at machine j. Let $\alpha^1_{i,j}$ denote the (corresponding) estimated arrival rate actually used by the policy. In our simulations, $\alpha^1_{i,j}$ is obtained using the following relation: $\alpha^1_{i,j} = \alpha_{i,j} \times (1 + E)$, where E is sampled from the uniform distribution $[-I, +I]$ and I is the maximum inaccuracy. Analogously defined, $\mu^1_{i,j}$ denotes the estimated execution rate of class i tasks at machine j used by the policy.

Figure 4.1 compares the LPAS_dec policy and the KPB policy (with $\overline{k} = 3$) in terms of their performance on System 4.A under different inaccuracy levels. We do not include the results for the MCT policy since it results in severe performance degradation or even system instability for low values of I. For example, using the MCT policy results in $\bar{W} = 6.61$ time-units when $I = 10\%$ and the system is unstable when $I = 30\%$. This experiment shows that the LPAS_dec policy has robustness advantages over the other policies which can be explained in part by the property that the solution to the allocation LP is inherently robust.

4.3.3 Realistic Architectures

To simulate more realistic scenarios, we use the data reported in [9, 15] which was collected by running benchmarking tools on an actual system (see Section 3.4.5). We refer to this system as System 4.B.

We simulate two configurations based on System 4.B (4.$B1$ and 4.$B2$). Both systems consist of $M = 300$ machines which are grouped into 15 groups. We simulate

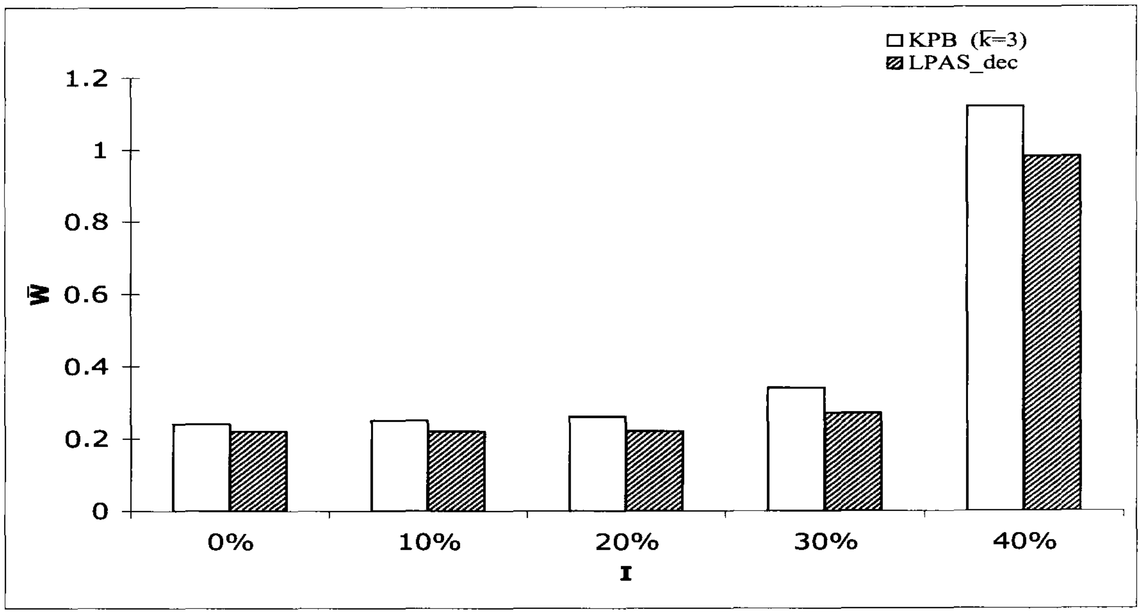

Figure 4.1: System 4.A simulation results under different values for the maximum inaccuracy

the execution of each system for two billion time-units.

To have dynamic CPU availabilities, the steady-state CPU availability for each machine changes randomly at regular rate-change events (time intervals between such events are exponentially distributed with mean 5 million time-units). At every rate-change event, each machine assumes the same parameters as one of the 15 machines of System 4.B listed in Table 4.14 in [15] (reproduced in Table 3.13) with equal probabilities. Let a_j be the steady-state CPU availability of a machine j. Thus, the execution rate for class i tasks at machine j is effectively $\mu_{i,j} \times a_j$. We assume that the load balancing policies use these estimated effective execution rates.

In System 4.$B1$, we assume that the machines of a group are identical in terms of their nominal computing powers. Each group has the same nominal computing power as one of the 15 machines of System 4.B. Furthermore, we assume that the nominal computing power of a machine depends only on the machine and is independent of the class of tasks being executed. System 4.$B1$ represents a system which is mainly

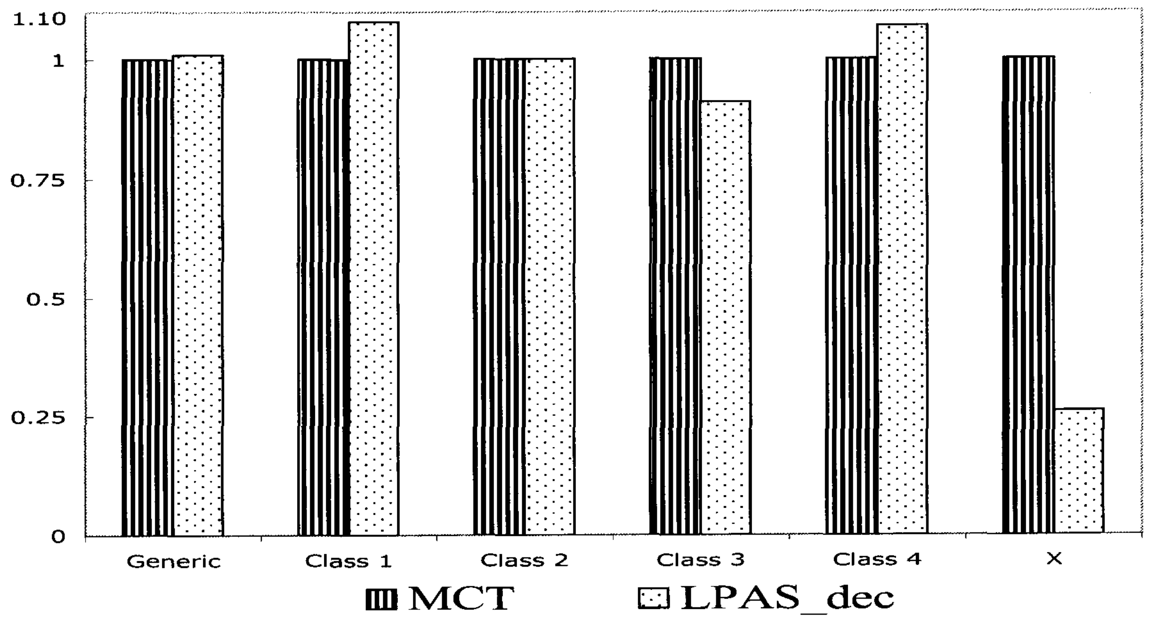

Figure 4.2: Relative average task completion times and number of message exchanges: System $4.B1$ under arrival rates α

used to execute a single application.

In System $4.B2$, we assume that each machine has a nominal computing power (on class i tasks) $P_{i,j}$ randomly chosen from $\{1, 1.125, 1.4375\}$ with equal probabilities. Thus, a machine can be fast executing some applications while, at the same time, slow executing other applications. System $4.B2$ represents a system which is mainly used to execute multiple applications with inherent heterogeneity.

Finally, we assume that there are $N = 4$ classes (or applications). We assume that $BaseTime_i$ = 8750, 17500, 35000, 50000, for $i = 1, \ldots, 4$, respectively. This information is enough to generate the matrix μ. Assuming $a_j = 1$, the mean execution time for a class i task at machine j can be computed as $BaseTime_i \times 1/P_{i,j}$.

Figures 4.2 and 4.3 show simulation results for Systems $4.B1$ and $4.B2$ under arrival rates $\alpha = [0.00457 \;\; 0.00229 \; 0.00114 \; 0.00080]$. For a machine j, we assume that $\alpha_{i,j} = \alpha_i/M$, $i = 1, \ldots, N$. In this section, we normalize the results with respect to the MCT policy and note that the accuracy of the generated confidence intervals is

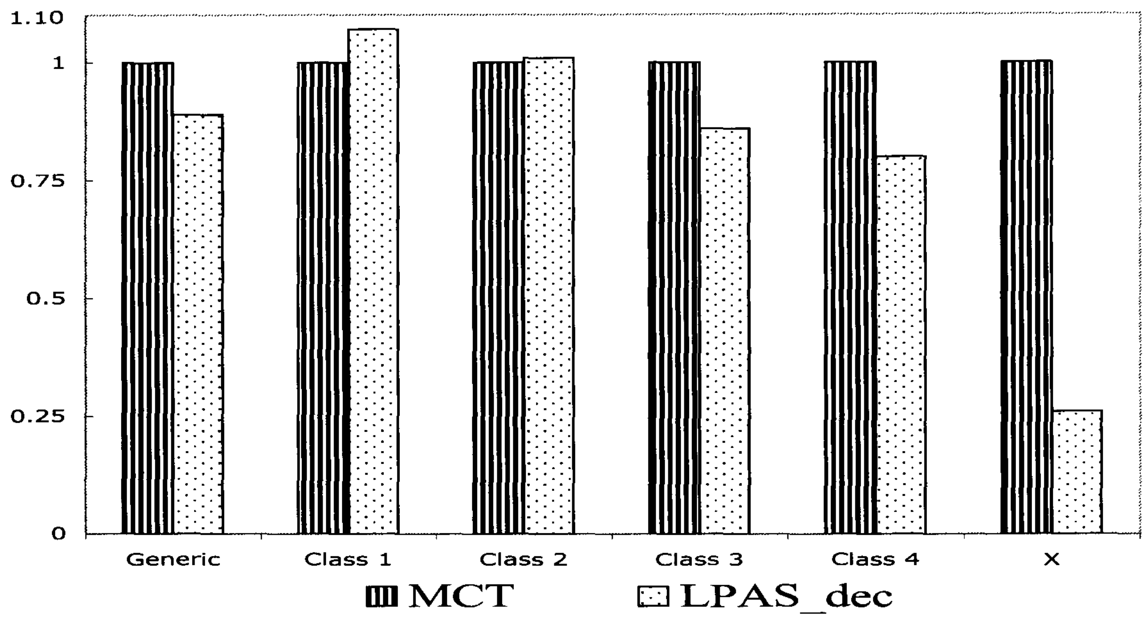

Figure 4.3: Relative average task completion times and number of message exchanges: System 4.$B2$ under arrival rates α

0.1% or less. The KPB policy is not included as it is difficult to find an optimal value for $\overline{k}$. These results indicate that the MCT policy achieves acceptable performance in systems with low task heterogeneity, such as System 4.$B1$. However, as the level of task heterogeneity increases (*e.g.* System 4.$B2$), the MCT policy results in performance degradation which gets worse as the load increases. The LPAS_dec policy is generally superior in highly heterogeneous and highly loaded systems. Of course, there is the added advantage of significant reduction in the overhead of state information exchange.

4.4 Summary

In our work, we have developed the LPAS_dec policy which utilizes the solution to an allocation LP. The policy takes into account information on heterogeneity resulting in improved performance. However, it requires dynamic knowledge on task arrival and machine execution rates. The information exchange mechanism used by the LPAS_dec policy is shown to dramatically cut down the communication overhead while being effective in updating the state information in a decentralized fashion.

Chapter 5

Conclusion

In this thesis, we have proposed several resource management policies for different HC systems. These policies utilize the solution to an allocation LP which maximizes the system capacity. Such policies require knowledge on both arrival and execution rates. Our simulation experiments show that these policies perform very competitively in highly heterogeneous systems.

There are several lines along which future research could proceed.

- We note that there has been little work done in characterizing actual HC system workloads. Hence, we believe that there is a need to develop a benchmark framework which characterizes actual workloads and can be used to compare different policies in terms of several performance metrics (for example, see the work of Li *et al.* [49]).

- The main issue addressed in our work is dealing with heterogeneity of HC systems within the context of resource management. This work does not include other factors such as communication delay, data transfer costs, heterogeneous network bandwidths, and network topologies. We believe that the basic framework presented here can be adapted to such settings.

- Resource management for HC systems is an emerging field of research. There are other kinds of HC systems that have not been discussed in this thesis. For example, a hierarchical grid can be constructed by using multiple computing clusters where each of these clusters has its own RMS (see Garonne *et al.* [32]). Another emerging kind of HC systems uses P2P (peer to peer) based technology for resource management (see Chakravarti *et al.* [17] and Kim *et al.* [41]) . Given the success of our linear programming based policies, it would be worthwhile to explore the potential of such an approach in developing resource management policies for these emerging HC systems.

Appendix A

Here, we apply the fluid limit methodology in proving several stability results. Our analysis will involve a formal limiting fluid model for the system. This is done by describing the system as a Markov process and performing a scaling in time and space that allows the use of law of large numbers results, leading to a deterministic model where the flow through the system is continuous (fluid) rather than discrete (tasks). The use of fluid model techniques for characterizing stability is a well established methodology: see, for example, the work of Chen [18], Chen and Yao [19], Dai [23, 24], and Dai and Meyn [25].

First, we define the system dynamics of the queueing network corresponding to our workload model. Class i tasks arrive via an arrival process with independent and identically distributed (i.i.d.) interarrival times $\{\xi_i(n)\}$ where $\alpha_i = 1/E[\xi_i(1)]$. Also, let $\eta_{i,j}(n)$ denote the execution time for the nth class i task executed at machine j, where $\mu_{i,j} = 1/E[\eta_{i,j}(1)]$ if machine j can execute class i tasks, and $\mu_{i,j} = 0$ otherwise. We assume that the sequence $\{\eta_{i,j}(n)\}$ is i.i.d. for each i and j. Let $A_i(t)$ be the residual interarrival time for class i tasks at time t. Let $Y_{i,j}(t)$ be the residual execution time for class i by machine j at time t.

Let $T_{i,j}(t)$ be the cumulative time that machine j has spent on class i tasks in $(0, t]$. Note that the functions $T_{i,j}(t)$ are determined by the mapping heuristic and the scheduling policy of machine j. Let $T_j(t) = \sum_{i=1}^{N} T_{i,j}(t)$ represent the cumulative time that machine j has been busy in $(0, t]$.

Define $W_{i,j}(t)$ as the cumulative amount of time that it takes machine j to execute class i tasks present in its queue at time t. Thus, $W_{i,j}(t)$ represents the class i workload of machine j at time t. We also define $Q_{i,j}(t)$ as the total queue length of class i tasks at machine j at time t. Let $\alpha_{i,j}(t)$ be the total number of class i tasks assigned by the mapping heuristic to machine j in $(0, t]$. With the definitions above, we are now able to give an expression for the evolution of $W_{i,j}(t)$, $i = 1, \ldots, N, j = 1, \ldots, M$,

$$W_{i,j}(t) = \sum_{n=1}^{Q_{i,j}(0)+\alpha_{i,j}(t)} \eta_{i,j}(n) - T_{i,j}(t).$$

Second, we construct a Markov process X for the system. For any of the mapping heuristics discussed in this paper,

$$X(t) := (W_{i,j}(t), A_i(t), Y_{i,j}(t) : i = 1, \ldots, N, j = 1, \ldots, M)$$

is a Markovian state evolving on

$$\mathbb{R}_+^{N\times M} \times \mathbb{R}_+^{N} \times \mathbb{R}_+^{N\times M}.$$

The process X may be shown to have the strong Markov property.

Let $w = \sum_{i=1}^{N}\sum_{j=1}^{M} W_{i,j}(0)$. Suppose that the function $(\overline{W}_{i,j}(t), \overline{T}_{i,j}(t) : i = 1, \ldots, N, j = 1, \ldots, M)$ is a limit point of the functions $(w^{-1}W_{i,j}(wt), w^{-1}T_{i,j}(wt) : i = 1, \ldots, N, j = 1, \ldots, M)$ when $w \to \infty$. We call $(\overline{W}_{i,j}(t), \overline{T}_{i,j}(t) : i = 1, \ldots, N, j = 1, \ldots, M)$ a fluid limit of the system.

We are now ready to describe the fluid model corresponding to our workload model. Let $(\overline{W}_{i,j}(t), \overline{T}_{i,j}(t) : i = 1, \ldots, N$ and $j = 1, \ldots, M)$ be a fluid limit for the system. Define $\alpha_{i,j}$ as $\lim_{t\to\infty} \frac{\alpha_{i,j}(t)}{t}$, $i = 1, \ldots, N, j = 1, \ldots, M$, assuming the limit exists (for the mapping heuristics we are concerned with, $\alpha_{i,j}$ does exist). For any mapping heuristic, every fluid limit satisfies the following set of conditions (for all $i = 1, \ldots, N$ and $j = 1, \ldots, M$):

(A.1) $\overline{W}_{i,j}(t) = \overline{W}_{i,j}(0) + \dfrac{\alpha_{i,j}t}{\mu_{i,j}} - \overline{T}_{i,j}(t)$;

(A.2) $\overline{W}_{i,j}(t) \geq 0$;

(A.3) $\overline{T}_{i,j}(0) = 0$ and $\overline{T}_{i,j}(\cdot)$ is nondecreasing;

(A.4) $0 \leq \sum_{i=1}^{N} \frac{d}{dt}\overline{T}_{i,j}(t) \leq 1.$

The derivatives above exist almost everywhere, as $\overline{T}_{i,j}(t)$ is Lipschitz for all i, j. From this point on, derivatives will be understood to be taken on the condition that they exist.

The conditions (A.1)-(A.4) do not completely specify the fluid limits, and there are other conditions on $\overline{T}_{i,j}(t)$. The complete set of conditions is known as the fluid model (see Theorem 2.3.2 of [23]). A fluid solution refers to any solution to the fluid model equations.

The fluid model is said to be stable if there exists a fixed time $t' > 0$ such that $\overline{W}_{i,j}(t) = 0$, $t > t'$, $i = 1, \ldots, N, j = 1, \ldots, M$, for any fluid solution. The fluid model is said to be (weakly) unstable if there exists a $t' > 0$ such that for every solution of the fluid model with $\sum_{i=1}^{N}\sum_{j=1}^{M}\overline{W}_{i,j}(0) = 0$, $\sum_{i=1}^{N}\sum_{j=1}^{M}\overline{W}_{i,j}(t') \neq 0$. Analyzing the stability region of the deterministic fluid model defined above allows us to characterize the maximum capacity of the actual system.

PROOF. [Theorem 2.2.1]
Consider the LP–Static heuristic. If $\lambda^* > 1$, we show that the LP–Static heuristic is guaranteed to stabilize the system. The LP–Static heuristic randomly maps tasks to machines according to probabilities $p_{i,j} = \frac{\delta^*_{i,j}\mu_{i,j}}{\lambda^*\alpha_i}$, $i = 1, \ldots, N, j = 1, \ldots, M$.

Let $W_j(t)$ denote the total workload at machine j at time t. Thus,

$$W_j(t) = \sum_{i=1}^{N} W_{i,j}(t).$$

Define $\overline{W}_j(t)$ as a limit point of the function $w^{-1}W_j(wt)$ as $w \to \infty$, $j = 1, \ldots, M$. Then,

$$\overline{W}_j(t) = \sum_{i=1}^{N} \overline{W}_{i,j}(t).$$

Note that if $\overline{W}_j(t) > 0$, then it must be true that $\frac{d}{dt}\overline{T}_j(t) = 1$. Hence, if $\overline{W}_j(t) > 0$, then

$$\begin{aligned}\frac{d}{dt}\overline{W}_j(t) &= \frac{\alpha_{1,j}}{\mu_{1,j}} + \cdots + \frac{\alpha_{N,j}}{\mu_{N,j}} - \frac{d}{dt}\overline{T}_{1,j}(t) - \cdots - \frac{d}{dt}\overline{T}_{N,j}(t)\\ &= \frac{\alpha_{1,j}}{\mu_{1,j}} + \cdots + \frac{\alpha_{N,j}}{\mu_{N,j}} - 1.\end{aligned}$$

Since using the LP–Static heuristic as a mapping heuristic results in $\alpha_{i,j} = \alpha_i p_{i,j} = \frac{\delta^*_{i,j}\mu_{i,j}}{\lambda^*}$, $i = 1, \ldots, N, j = 1, \ldots, M$, it must be true that

$$\begin{aligned}\frac{d}{dt}\overline{W}_j(t) &= \frac{\delta^*_{1,j}}{\lambda^*} + \cdots + \frac{\delta^*_{N,j}}{\lambda^*} - 1\\ &= \frac{\sum_{j=1}^{M}\delta^*_{i,j}}{\lambda^*} - 1\\ &< 0 \quad \text{since } \sum_{j=1}^{M}\delta^*_{i,j} \leq 1 \text{ and } \lambda^* > 1.\end{aligned}$$

Thus, if $\overline{W}_j(t) > 0$, then $\frac{d}{dt}\overline{W}_j(t) < 0$ which implies that there exists a fixed time $t' > 0$ such that $\overline{W}_j(t) = 0$, and hence $\overline{W}_{i,j}(t) = 0$, $i = 1, \ldots, N, j = 1, \ldots, M$, for all $t > t'$. Hence, the fluid model is stable and the result follows from Theorem 4.2 in [24].

PROOF. [Theorem 2.2.2]
Assume that the system can be stabilized. Hence, the corresponding fluid model is stable *i.e.*, there exists a fixed time $t' > 0$ such that $\overline{W}_{i,j}(t) = 0$, $t > t'$, $i = 1, \ldots, N, j = 1, \ldots, M$, for any fluid solution. Choose $s > t'$. Then, $\frac{d}{dt}\overline{W}_{i,j}(s) = 0$, $i = 1, \ldots, N, j = 1, \ldots, M$. Also, let $\frac{d}{dt}\overline{T}_{i,j}(s) = \delta_{i,j}$, $i = 1, \ldots, N, j = 1, \ldots, M$. Condition (A.1) implies (after taking the derivative of both terms and substituting s for t),

$$\frac{\alpha_{i,j}}{\mu_{i,j}} - \delta_{i,j} = 0, \qquad \text{for all } i = 1, \ldots, N, j = 1, \ldots, M.$$

Thus,

$$0 = \alpha_{i,j} - \delta_{i,j}\mu_{i,j}, \qquad \text{for all } i = 1, \ldots, N, j = 1, \ldots, M.$$

Summing over j,

$$0 = \alpha_i - \sum_{j=1}^{M} \delta_{i,j}\mu_{i,j}, \qquad \text{for all } i = 1, \ldots, N.$$

Thus, the following constraints hold ((A.6) and (A.7) follow from (A.4)):

(A.5)

$$\sum_{j=1}^{M} \delta_{i,j}\mu_{i,j} \geq \alpha_i, \qquad \text{for all } i = 1, \ldots, N,$$

(A.6)

$$\sum_{i=1}^{N} \delta_{i,j} \leq 1, \qquad \text{for all } j = 1, \ldots, M,$$

(A.7)

$$\delta_{i,j} \geq 0, \qquad \text{for all } i = 1, \ldots, N, \text{ and } j = 1, \ldots, M.$$

Thus, (A.5)-(A.7) provide a feasible solution for the allocation LP (2.1)-(2.3) with $\lambda^* = 1$ contradicting the assumption that $\lambda^* < 1$. Hence, the fluid model is weakly unstable and by Theorem 2.5.1 of [23], the system can not be stabilized.

PROOF. [Theorem 2.4.1]

Using the Guided-LPAS heuristic (introduced in Section 2.4), we can show that

(A.8) $\pi_{i,j}\alpha_i(t) - M + 1 - \sum_{j' \neq j} C_{i,j'}\sqrt{t} \leq \alpha_{i,j}(t) < \pi_{i,j}\alpha_i(t) + C_{i,j}\sqrt{t} + 1.$

First, let us show that

(A.9) $\alpha_{i,j}(t) < \pi_{i,j}\alpha_i(t) + C_{i,j}\sqrt{t} + 1.$

Assume that the Guided-LPAS heuristic maps an arrival of class i at time t to machine j. Since the arriving task was mapped onto machine j, it must be true from the definition of the Guided-LPAS heuristic (see condition (iii)) that $\alpha_{i,j}(t^-) < \pi_{i,j}\alpha_i(t) + C_{i,j}\sqrt{t}$. It then follows that $\alpha_{i,j}(t) = \alpha_{i,j}(t^-) + 1 < \pi_{i,j}\alpha_i(t) + C_{i,j}\sqrt{t} + 1$, proving (A.9).

Second, we show

$$\text{(A.10)} \quad \pi_{i,j}\alpha_i(t) - M + 1 - \sum_{j' \neq j} C_{i,j'}\sqrt{t} \leq \alpha_{i,j}(t).$$

Consider a machine j. Clearly it is assigned the following number of class i tasks:

$$\alpha_{i,j}(t) = \alpha_i(t) - \sum_{j' \neq j} \alpha_{i,j'}(t).$$

where $j' \in \{1, \ldots, M\}$. Using the Guided-LPAS heuristic, (A.9) holds and it follows that

$$\begin{aligned} \alpha_{i,j}(t) &= \alpha_i(t) - \sum_{j' \neq j} \alpha_{i,j'}(t) \\ &\geq \alpha_i(t) - \sum_{j' \neq j} (\pi_{i,j'}\alpha_i(t) + C_{i,j'}\sqrt{t} + 1) \\ &= \alpha_i(t) - \sum_{j' \neq j} \pi_{i,j'}\alpha_i(t) - \sum_{j' \neq j} C_{i,j'}\sqrt{t} - (M-1) \\ &= \pi_{i,j}\alpha_i(t) - M + 1 - \sum_{j' \neq j} C_{i,j'}\sqrt{t}. \end{aligned}$$

This proves (A.10).

From (A.8), it follows that $\alpha_{i,j} = \alpha_i\pi_{i,j} = \frac{\delta^*_{i,j}\mu_{i,j}}{\lambda^*}$, $i = 1, \ldots, N, j = 1, \ldots, M$. Thus, the results follow as before (see the proof of Theorem 2.2.1).

PROOF. [Theorem 2.4.2]
Assume that the Guided-LPAS$-2/k$ heuristic maps an arrival of class i at time t to machine j. Since the arriving task was mapped onto machine j, it must be true from the definition of the Guided-LPAS$-2/k$ heuristic (see the definition of $T_i(t)$ of the heuristic) that $\alpha_{i,j}(t^-) < \pi_{i,j}\alpha_i(t) + C_{i,j}\sqrt{t}$. This is the key to the proof of (A.8) and the results follow as before (see the proof of Theorem 2.4.1).

Bibliography

[1] I. Al-Azzoni and D. G. Down. Linear Programming Based Affinity Scheduling for Desktop Grids, working paper.

[2] I. Al-Azzoni and D. G. Down. Decentralized load balancing for heterogeneous grids. In *Proceedings of the 1st International Conference on Future Computational Paradigms and Applications (Future Computing 2009)*, to appear.

[3] I. Al-Azzoni and D. G. Down. Linear programming based affinity scheduling for heterogeneous computing systems. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 105–111, 2007.

[4] I. Al-Azzoni and D. G. Down. Dynamic scheduling for heterogeneous Desktop Grids. In *Proceedings of the 9th International Conference on Grid Computing*, pages 136–143, 2008.

[5] I. Al-Azzoni and D. G. Down. Linear programming based affinity scheduling of independent tasks on heterogeneous computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 19(12):1671–1682, 2008.

[6] S. Ali, A. A. Maciejewski, H. J. Siegel, and J.-K. Kim. Measuring the robustness of a resource allocation. *IEEE Transactions on Parallel and Distributed Systems*, 15(7):630–641, 2004.

[7] S. Andradóttir, H. Ayhan, and D. G. Down. Dynamic server allocation for queueing networks with flexible servers. *Operations Research*, 51(6):952–968, 2003.

[8] S. Andradóttir, H. Ayhan, and D. G. Down. Compensating for failures with flexible servers. *Operations Research*, 55(4):753–768, 2007.

[9] C. Anglano, J. Brevik, M. Canonico, D. Nurmi, and R. Wolski. Fault-aware scheduling for Bag-of-Tasks applications on Desktop Grids. In *Proceedings of the 7th International Conference on Grid Computing*, pages 56–63, 2006.

[10] P. S. Ansell, K. D. Glazebrook, and C. Kirkbride. Generalised 'join the shortest queue' policies for the dynamic routing of jobs to multiclass queues. *Journal of the Operational Research Society*, 54:379–389, 2003.

[11] R. Armstrong. Investigation of effect of different run-time distributions on SmartNet performance. Master's thesis, Naval Postgraduate School, 1997.

[12] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4.1. Innovative Computing Dept. Technical Report ICL-UT-02-05, University of Tennessee, Knoxville, TN, June 2002.

[13] M. Arora, S. K. Das, and R. Biswas. A de-centralized scheduling and load balancing algorithm for heterogeneous grid environments. In *Proceedings of the International Conference on Parallel Processing Workshops*, pages 499–505, 2002.

[14] T. D. Braun, H. J. Siegel, and A. A. Maciejewski. Heterogeneous computing: Goals, methods, and open problems. In *Proceedings of the 8th International Conference on High Performance Computing*, pages 307–320, 2001.

[15] M. Canonico. Scheduling Algorithms for Bag-of-Tasks Applications on Fault-Prone Desktop Grids. PhD thesis, University of Turin, 2006.

[16] H. Casanova, D. Zagorodnov, F. Berman, and A. Legrand. Heuristics for scheduling parameter sweep applications in grid environments. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 349–363, 2000.

[17] A. J. Chakravarti, G. Baumgartner, and M. Lauria. The organic grid: self-organizing computation on a peer-to-peer network. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 35(3):373–384, 2005.

[18] H. Chen. Fluid approximations and stability of multiclass queueing networks: Work-conserving disciplines. *Annals of Applied Probability*, 5:637–655, 1995.

[19] H. Chen and D. Yao. *Fundamentals of Queueing Networks: Performance, Asymptotics and Optimization.* Springer-Verlag, 2001.

[20] S. Choi, H. Kim, E. Byun, M. Baik, S. Kim, C. Park, and C. Hwang. Characterizing and classifying desktop grid. In *Proceedings of the 7th International Symposium on Cluster Computing and the Grid*, pages 743–748, 2007.

[21] S. Choi, H. Kim, E. Byun, and C. Hwang. A taxonomy of desktop grid systems focusing on scheduling. Technical Report KU-CSE-2006-1120-01, Department of Computer Science and Engeering, Korea University, November 2006.

[22] Condor. "http://www.cs.wisc.edu/condor/".

[23] J. G. Dai. *Stability of Fluid and Stochastic Processing Networks.* Publication No. 9, 1999. Centre for Mathematical Physics and Stochastics. http://www.maphysto.dk/.

[24] J. G. Dai. On positive Harris recurrence of multiclass queueing networks: a unified approach via fluid limit models. *Annals of Applied Probability*, 5:49–77, 1995.

[25] J. G. Dai and S. Meyn. Stability and convergence of moments for multiclass queueing networks via fluid limit models. *IEEE Transactions on Automatic Control*, 40:1889–1904, 1995.

[26] P. Domingues, A. Andrzejak, and L. Silva. Scheduling for fast turnaround time on institutional desktop grid. Technical Report TR-0027, CoreGRID, January 2006.

[27] P. Domingues, P. Marques, and L. Silva. DGSchedSim: A trace-driven simulator to evaluate scheduling algorithms for desktop grid environments. In *Proceedings of the 14th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 83–90, 2006.

[28] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3):200–222, 2001.

[29] H. Franke, J. Jann, J. E. Moreira, P. Pattnaik, and M. A. Jette. An evaluation of parallel job scheduling for ASCI Blue-Pacific. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, pages 11–18, 1999.

[30] R. Freund, M. Gherrity, S. Ambrosius, M. Campbell, M. Halderman, D. Hensgen, E. Keith, T. Kidd, M. Kussow, J. D. Lima, F. Mirabile, L. Moore, B. Rust, and H. J. Siegel. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In *Proceedings of the 7th Heterogeneous Computing Workshop*, pages 184–199, 1998.

[31] R. Freund, T. Kidd, and L. Moore. SmartNet: a scheduling framework for heterogeneous computing. In *Proceedings of the 2nd International Symposium on Parallel Architectures, Algorithms and Networks*, pages 514–521, 1996.

[32] V. Garonne, A. Tsaregorodtsev, and E. Caron. A study of meta-scheduling architectures for high throughput computing: Pull versus push. In *Proceedings*

of the 4th International Symposium on Parallel and Distributed Computing, pages 226–233, 2005.

[33] K. D. Glazebrook and C. Kirkbride. Dynamic routing to heterogeneous collections of unreliable servers. *Queueing Systems: Theory and Applications*, 55(1):9–25, 2007.

[34] D. Gu, L. Yang, and L. R. Welch. A predictive, decentralized load balancing approach. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium*, 2005.

[35] Y.-T. He. *Exploiting Limited Customer Choice and Server Flexibility*. PhD thesis, McMaster University, 2007.

[36] Y.-T. He, I. Al-Azzoni, and D. G. Down. MARO - MinDrift affinity routing for resource management in heterogeneous computing systems. In *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research*, pages 71–85, 2007.

[37] ILOG CPLEX. "http://www.ilog.com/products/cplex/".

[38] A. Iosup, O. Sonmez, S. Anoep, and D. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, pages 97–108, 2008.

[39] A. Iosup, O. Sonmez, S. Anoep, and D. Epema. The performance of bags-of-tasks in large-scale distributed systems. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, pages 97–108, 2008.

[40] J.-K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. D. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S. S. Yellampalli. Dynamically mapping tasks with

priorities and multiple deadlines in a heterogeneous environment. *Journal of Parallel and Distributed Computing*, 67(2):154–169, 2007.

[41] J.-S. Kim, B. Nam, P. Keleher, M. Marsh, B. Bhattacharjee, and A. Sussman. Trade-offs in matching jobs and balancing load for distributed desktop grids. *Future Generation Computer Systems*, 24(5):415–424, 2008.

[42] M. Kokaly, I. Al-Azzoni, and D. G. Down. MGST: a framework for the performance evaluation of Desktop Grids. In *Proceedings of the 24th International Parallel and Distributed Processing Symposium*, pages 1–8, 2009.

[43] D. Kondo, A. Andrzejak, and D. P. Anderson. On correlated availability in Internet-distributed systems. In *Proceedings of the 9th International Conference on Grid Computing*, pages 276–283, 2008.

[44] D. Kondo, A. A. Chien, and H. Casanova. Resource management for rapid application turnaround on enterprise desktop grids. In *Proceedings of the ACM/IEEE Conference on Supercomputing*, 2004.

[45] D. Kondo, G. Fedak, F. Cappello, A. A. Chien, and H. Casanova. Characterizing resource availability in enterprise desktop grids. *Future Generation Computer Systems*, 23(7):888–903, 2007.

[46] L. Kontothanassis and D. Goddeau. Profile driven scheduling for a heterogeneous server cluster. In *Proceedings of the 34th International Conference on Parallel Processing Workshops*, pages 336–345, 2005.

[47] I. Legrand, H. Newman, R. Voicu, C. Cirstoiu, C. Grigoras, M. Toarta, and C. Dobre. MonALISA: an agent based, dynamic service system to monitor, control and optimize grid based applications. In *Proceedings of the International Conference on Computing in High Energy and Nuclear Physics*, 2004.

[48] H. Li and R. Buyya. Model-driven simulation of grid scheduling strategies. In *Proceedings of the 3rd International Conference on e-Science and Grid Computing*, pages 287–294, 2007.

[49] H. Li, D. Groep, and L. Wolters. Workload characteristics of a multi-cluster supercomputer. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, pages 176–193. Springer Verlag, 2004. Lect. Notes Comput. Sci. vol. 3277.

[50] K. Lu, R. Subrata, and A. Y. Zomaya. On the performance-driven load distribution for heterogeneous computational grids. *Journal of Computer and System Sciences*, 73(8):1191–1206, 2007.

[51] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund. Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems. In *Proceedings of the 8th Heterogeneous Computing Workshop*, pages 30–44, 1999.

[52] A. Mandelbaum and A. L. Stolyar. Scheduling flexible servers with convex delay costs: Heavy-traffic optimality of the generalized cμ-rule. *Operations Research*, 52(6):836–855, 2004.

[53] A. M. Mehta, J. Smith, H. J. Siegel, A. A. Maciejewski, A. Jayaseelan, and B. Ye. Dynamic resource allocation heuristics that manage tradeoff between makespan and robustness. *Journal of Supercomputing*, 42(1):33–58, 2007.

[54] M. Mitzenmacher. How useful is old information? *IEEE Transactions on Parallel and Distributed Systems*, 11(1):6–20, 2000.

[55] M. Mitzenmacher. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(10):1094–1104, 2001.

[56] D. Nurmi, J. Brevik, and R. Wolski. Modeling machine availability in enterprise and wide-area distributed computing environments. In *Proceedings of the 11th International Euro-Par Conference*, pages 432–441, 2005.

[57] I. Rao and E.-N. Huh. A probabilistic and adaptive scheduling algorithm using system-generated predictions for inter-grid resource sharing. *Journal of Supercomputing*, 45(2):185–204, 2008.

[58] X. Ren, S. Lee, R. Eigenmann, and S. Bagchi. Prediction of resource availability in fine-grained cycle sharing systems empirical evaluation. *Journal of Grid Computing*, 5(2):173–195, 2007.

[59] B. Rood and M. J. Lewis. Multi-state grid resource availability characterization. In *Proceedings of the 8th International Conference on Grid Computing*, pages 42–49, 2007.

[60] B. Rood and M. J. Lewis. Scheduling on the Grid via multi-state resource availability prediction. In *Proceedings of the 9th International Conference on Grid Computing*, pages 126–135, 2008.

[61] SETI@home. “http://setiathome.berkeley.edu/”.

[62] R. Shah, B. Veeravalli, and M. Misra. On the design of adaptive and decentralized load balancing algorithms with load estimation for computational grid environments. *IEEE Transactions on Parallel and Distributed Systems*, 18(12):1675–1686, 2007.

[63] A. Sharifnia. Instability of the join-the-shortest-queue and FCFS policies in queuing systems and their stabilization. *Operations Research*, 45(2):309–314, 1997.

[64] V. Shestak, J. Smith, H. J. Siegel, and A. A. Maciejewski. A stochastic approach to measuring the robustness of resource allocations in distributed systems. In

Proceedings of the International Conference on Parallel Processing, pages 459–470, 2006.

[65] J. Smith, L. Briceno, A. A. Maciejewski, H. J. Siegel, T. Renner, V. Shestak, J. Ladd, A. Sutton, D. Janovy, S. Govindasamy, A. Alqudah, R. Dewri, and P. Prakash. Measuring the robustness of resource allocations in a stochastic dynamic environment. In *Proceedings of the International Parallel and Distributed Processing Symposium*, 2007.

[66] T. Sterling, E. Lusk, and W. Gropp, editors. *Beowulf Cluster Computing with Linux.* MIT Press, Cambridge, MA, USA, 2003.

[67] A. Stolyar. Optimal routing in output-queued flexible server systems. *Probability in the Engineering and Information Sciences*, 19(2):141–189, 2005.

[68] K. Wasserman, G. Michailidis, and N. Bambos. Optimal processor allocation to differentiated job flows. *Performance Evaluation*, 63(1):1–14, 2006.

[69] P. Werstein, H. Situ, and Z. Huang. Load balancing in a cluster computer. In *Proceedings of the 7th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 569–577, 2006.

[70] R. Wolski, N. Spring, and J. Hayes. Predicting the CPU availability of time-shared Unix systems on the computational grid. *Cluster Computing*, 3(4):293–301, 2000.

[71] R. Wolski, N. T. Spring, and J. Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, 15(5-6):757–768, 1999.

[72] L. Yang, J. M. Schopf, and I. Foster. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. In *Proceedings of the ACM/IEEE conference on Supercomputing*, 2003.

[73] Y. Zhang and Y. Inoguchi. Influence of inaccurate performance prediction on task scheduling in a grid environment. *IEICE - Transactions on Information and Systems*, E89-D(2):479–486, 2006.

5359 03